Sp cification Volum 1

Specification of the Bluetooth System

Wireless connections made easy

Core

Bluetooth™

v1.0 B
December 1st 1999

BLUETOOTH DOC	Date / Day-Month-Year 01 Dec 99	N.B.	Document No. 1.C.47/1.0 B
Responsible	e-mail address		Status

Bluetooth.

Specification of the Bluetooth System

Version 1.0 B

Revision History

The Revision History is shown in Appendix I on page 868

Contributors

The persons who contributed to this specification are listed in Appendix II on page 879.

Web Site

This specification can also be found on the Bluetooth website:
http://www.bluetooth.com

Disclaimer and copyright notice

Bluetooth.

MASTER TABLE OF CONTENTS

For the Bluetooth Profiles, *See Volume 2.*

Part A **Volume 1 (1:2)**

RADIO SPECIFICATION

Part B **Volume 1 (1:2)**

BASEBAND SPECIFICATION

Part C **Volume 1 (1:2)**

LINK MANAGER PROTOCOL

Contents **[C] 187**

Part D **Volume 1 (1:2)**

LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION

Contents **[D] 247**

Bluetooth.

Part E **Volume 1 (1:2)**

SERVICE DISCOVERY PROTOCOL (SDP)

Contents .... [E] 325

Part F:1 **Volume 1 (1:2)**

RFCOMM WITH TS 07.10

Contents .... [F:1] 387

Part F:2 **Volume 1 (1:2)**

IrDA INTEROPERABILITY

Contents .... [F:2] 413

Bluetooth.

Part F:3 **Volume 1 (1:2)**

TELEPHONY CONTROL PROTOCOL SPECIFICATION

Part F:4 **Volume 1 (1:2)**

INTEROPERABILITY REQUIREMENTS FOR BLUETOOTH AS A WAP BEARER

Part H:1 **Volume 1 (2:2)**

HOST CONTROLLER INTERFACE FUNCTIONAL SPECIFICATION

Part H:2 **Volume 1 (2:2)**

HCI USB TRANSPORT LAYER

Part H:3 **Volume 1 (2:2)**

HCI RS232 TRANSPORT LAYER

Bluetooth.

Part H:4 **Volume 1 (2:2)**

HCI UART TRANSPORT LAYER

Part I:1 **Volume 1 (2:2)**

BLUETOOTH TEST MODE

Part I:2 **Volume 1 (2:2)**

BLUETOOTH COMPLIANCE REQUIREMENTS

Bluetooth.

Part I:3 **Volume 1 (2:2)**

TEST CONTROL INTERFACE

Part K **Profiles - see Volume 2**

Appendix I **Volume 1 (2:2)**

Appendix II **Volume 1 (2:2)**

Appendix III **Volume 1 (2:2)**

Appendix IV **Volume 1 (2:2)**

SAMPLE DATA

·Bluetooth.

Appendix V **Volum 1 (2:2)**

BLUETOOTH AUDIO

Appendix VI **Volume 1 (2:2)**

BASEBAND TIMERS

Appendix VII **Volume 1 (2:2)**

OPTIONAL PAGING SCHEMES

Appendix VIII **Volume 1 (2:2)**

BLUETOOTH ASSIGNED NUMBERS

Appendix IX **Volume 1 (2:2)**

MESSAGE SEQUENCE CHARTS

Part A

RADIO SPECIFICATION

CONTENTS

1 SCOPE

The Bluetooth transceiver is operating in the 2.4 GHz ISM band. This specification defines the requirements for a Bluetooth transceiver operating in this unlicensed band.

Requirements are defined for two reasons:

- Provide compatibility between the radios used in the system
- Define the quality of the system

The Bluetooth transceiver shall fulfil the stated requirements under the operating conditions specified in Appendix A and Appendix B. The Radio parameters must be measured according to the methods described in the RF Test Specification.

This specification is based on the established regulations for Europe, Japan and North America. The standard documents listed below are only for information, and are subject to change or revision at any time.

Europe (except France and Spain):

Approval Standards: European Telecommunications Standards Institute, ETSI
Documents: ETS 300-328, ETS 300-826
Approval Authority: National Type Approval Authorities

France:

Approval Standards: La Reglementation en France por les Equipements fonctionnant dans la bande de frequences 2.4 GHz "RLAN-Radio Local Area Network"
Documents: SP/DGPT/ATAS/23, ETS 300-328, ETS 300-826
Approval Authority: Direction Generale des Postes et Telecommunications
Note: A new R&TTE EU Directive will be in effect by March 2000, with consequent effects on the manufacturer's declaration of conformity and free circulation of products within the EU.

Spain:

Approval Standards: Supplemento Del Numero 164 Del Boletin Oficial Del Estado (Published 10 July 91, Revised 25 June 93)
Documents: ETS 300-328, ETS 300-826
Approval Authority: Cuadro Nacional De Atribucion De Frecuesias

Japan:

Approval Standards: Association of Radio Industries and Businesses, ARIB
Documents: RCR STD-33A
Approval Authority: Ministry of Post and Telecommunications, MPT
Note: The Japanese rules are in revision. Decisions on the revision will take place in Q2 1999.

North Americas:

Approval Standards: Federal Communications Commission, FCC, USA
Documents: CFR47, Part 15, Sections 15.205, 15.209, 15.247

Approval Standards: Industry Canada, IC, Canada
Documents: GL36

Approval Authority: FCC (USA), Industry Canada (Canada)

2 FREQUENCY BANDS AND CHANNEL ARRANGEMENT

The Bluetooth system is operating in the 2.4 GHz ISM (Industrial Scientific Medicine) band. In a vast majority of countries around the world the range of this frequency band is 2400 - 2483.5 MHz. Some countries have however national limitations in the frequency range. In order to comply with these national limitations, special frequency hopping algorithms have been specified for these countries. It should be noted that products implementing the reduced frequency band will not work with products implementing the full band. The products implementing the reduced frequency band must therefore be considered as local versions for a single market. The Bluetooth SIG has launched a campaign to overcome these difficulties and reach total harmonization of the frequency band.

Geography	Regulatory Range	RF Channels
USA, Europe and most other countries[1]	2.400-2.4835 GHz	f=2402+k MHz, k=0,...,78
Spain[2]	2.445-2.475 GHz	f=2449+k MHz, k=0,...,22
France[3]	2.4465-2.4835 GHz	f=2454+k MHz, k=0,...,22

Table 2.1: Operating frequency bands

Note 1. Japan, the MPT announced at the beginning of October 1999 that the Japanese frequency band would be extended to 2400-2483.5 MHz, effective immediately. Testing of devices by TELEC may however need some time to change. The previously specified special frequency-hopping algorithm covering 2471-2497 MHz remains as an option.

Note 2. There is a proposal in Spain to extend the national frequency band to 2403-2483.5 MHz. The Bluetooth SIG has approached the authorities in Spain to get a full harmonization. The outcome is expected by the beginning of year 2000.

Note 3. The Bluetooth SIG has established good contacts with the French authorities and are closely following the development of harmonization.

Channel spacing is 1 MHz. In order to comply with out-of-band regulations in each country, a guard band is used at the lower and upper band edge.

Geography	Lower Guard Band	Upper Guard Band
USA	2 MHz	3.5 MHz
Europe (except Spain and France)	2 MHz	3.5 MHz
Spain	4 MHz	26 MHz
France	7.5 MHz	7.5 MHz
Japan	2 MHz	2 MHz

Table 2.2: Guard Bands

3 TRANSMITTER CHARACTERISTICS

The requirements stated in this section are given as power levels at the antenna connector of the equipment. If the equipment does not have a connector, a reference antenna with 0 dBi gain is assumed.

Due to difficulty in measurement accuracy in radiated measurements, it is preferred that systems with an integral antenna provide a temporary antenna connector during type approval.

If transmitting antennas of directional gain greater than 0 dBi are used, the applicable paragraphs in ETSI 300 328 and FCC part 15 must be compensated for.

The equipment is classified into three power classes.

Power Class	Maximum Output Power (Pmax)	Nominal Output Power	Minimum Output Power[1]	Power Control
1	100 mW (20 dBm)	N/A	1 mW (0 dBm)	Pmin<+4 dBm to Pmax Optional: Pmin[2] to Pmax
2	2.5 mW (4 dBm)	1 mW (0 dBm)	0.25 mW (-6 dBm)	Optional: Pmin[2] to Pmax
3	1 mW (0 dBm)	N/A	N/A	Optional: Pmin[2] to Pmax

Table 3.1: Power classes

Note 1. Minimum output power at maximum power setting.

Note 2. The lower power limit Pmin<-30dBm is suggested but is not mandatory, and may be chosen according to application needs.

A power control is required for power class 1 equipment. The power control is used for limiting the transmitted power over 0 dBm. Power control capability under 0 dBm is optional and could be used for optimizing the power consumption and overall interference level. The power steps shall form a monotonic sequence, with a maximum step size of 8 dB and a minimum step size of 2 dB. A class 1 equipment with a maximum transmit power of +20 must be able to control its transmit power down to 4 dBm or less.

Equipment with power control capability optimizes the output power in a link with LMP commands (see Link Manager Protocol). It is done by measuring RSSI and report back if the power should be increased or decreased.

3.1 MODULATION CHARACTERISTICS

The Modulation is GFSK (Gaussian Frequency Shift Keying) with a BT=0.5. The Modulation index must be between 0.28 and 0.35. A binary one is represented by a positive frequency deviation, and a binary zero is represented by a negative frequency deviation. The symbol timing shall be better than ±20 ppm.

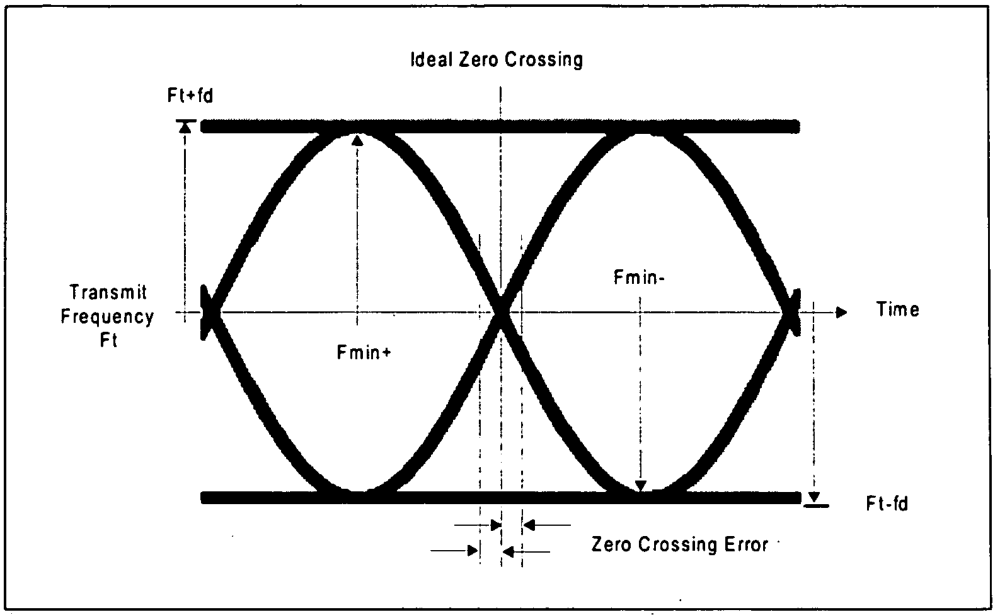


Figure 3.1: Figure 3-1 Actual transmit modulation.

For each transmit channel, the minimum frequency deviation (Fmin = the lesser of {Fmin+, Fmin-}) which corresponds to 1010 sequence shall be no smaller than ±80% of the frequency deviation (fd) which corresponds to a 00001111 sequence.
In addition, the minimum deviation shall never be smaller than 115 kHz.

The zero crossing error is the time difference between the ideal symbol period and the measured crossing time. This shall be less than ± 1/8 of a symbol period.

3.2 SPURIOUS EMISSIONS

The spurious emission, in-band and out-of-band, is measured with a frequency hopping transmitter hopping on a single frequency; this means that the synthesizer must change frequency between receive slot and transmit slot, but always returns to the same transmit frequency.

For the USA, FCC parts 15.247, 15.249, 15.205 and 15.209 are applicable regulations. For Japan, RCR STD-33 applies and, for Europe, ETSI 300 328.

3.2.1 In-band Spurious Emission

Within the ISM band the transmitter shall pass a spectrum mask, given in Table 3.2. The spectrum must comply with the FCC's 20-dB bandwidth definition stated below, and should be measured accordingly. In addition to the FCC requirement an adjacent channel power on adjacent channels with a difference in channel number of two or greater an adjacent channel power is defined. This adjacent channel power is defined as the sum of the measured power in a 1 MHz channel. The transmitted power shall be measured in a 100 kHz bandwidth using maximum hold. The transmitter is transmitting on channel M and the adjacent channel power is measured on channel number N. The transmitter is sending a pseudo random data pattern throughout the test.

Frequency offset	Transmit Power
± 550 kHz	-20 dBc
$\lvert M-N \rvert = 2$	-20 dBm
$\lvert M-N \rvert \geq 3$	-40 dBm

Table 3.2: Transmit Spectrum mask.

Note: If the output power is less than 0dBm then, wherever appropriate, the FCC's 20 dB relative requirement overrules the absolute adjacent channel power requirement stated in the above table.

"In any 100 kHz bandwidth outside the frequency band in which the spread spectrum intentional radiator is operating, the radio frequency power that is produced by the intentional radiator shall be at least 20 dB below that in the 100 kHz bandwidth within the band that contains the highest level of the desired power, based on either an RF conducted or a radiated measurement. Attenuation below the general limits specified in § 15.209(a) is not required. In addition, radiated emissions which fall in the restricted bands, as defined in § 15.205(a), must also comply with the radiated emission limits specified in § 15.209(a) (see § 15.205(c))."

FCC Part 15.247c

Exceptions are allowed in up to three bands of 1 MHz width centered on a frequency which is an integer multiple of 1 MHz. They must, however, comply with an absolute value of –20 dBm.

3.2.2 Out-of-Band Spurious Emission

The measured power should be measured in a 100 kHz bandwidth.

Frequency Band	Operation mode	Idle mode
30 MHz - 1 GHz	-36 dBm	-57 dBm
1 GHz – 12.75 GHz	-30 dBm	-47 dBm
1.8 GHz – 1.9 GHz	-47 dBm	-47 dBm
5.15 GHz – 5.3 GHz	-47 dBm	-47 dBm

Table 3.3: Out-of-band spurious emission requirement

3.3 RADIO FREQUENCY TOLERANCE

The transmitted initial center frequency accuracy must be ±75 kHz from F_c. The initial frequency accuracy is defined as being the frequency accuracy before any information is transmitted. Note that the frequency drift requirement is not included in the ±75 kHz.

The transmitter center frequency drift in a packet is specified in Table 3.4. The different packets are defined in the Baseband Specification.

Type of Packet	Frequency Drift
One-slot packet	±25 kHz
Three-slot packet	±40 kHz
Five-slot packet	±40 kHz
Maximum drift rate 1)	400 Hz/µs

Table 3.4: Frequency drift in a package

Note 1. The maximum drift rate is allowed anywhere in a packet.

4 RECEIVER CHARACTERISTICS

In order to measure the bit error rate performance; the equipment must have a "loop back" facility. The equipment sends back the decoded information. This facility is specified in the Test Mode Specification.

The reference sensitivity level referred to in this chapter equals -70 dBm.

4.1 ACTUAL SENSITIVITY LEVEL

The actual sensitivity level is defined as the input level for which a raw bit error rate (BER) of 0.1% is met. The requirement for a Bluetooth receiver is an actual sensitivity level of –70 dBm or better. The receiver must achieve the –70 dBm sensitivity level with any Bluetooth transmitter compliant to the transmitter specification specified in Section 3 on page 20.

4.2 INTERFERENCE PERFORMANCE

The interference performance on Co-channel and adjacent 1 MHz and 2 MHz are measured with the wanted signal 10 dB over the reference sensitivity level. On all other frequencies the wanted signal shall be 3 dB over the reference sensitivity level. Should the frequency of an interfering signal lie outside of the band 2400-2497 MHz, the out-of-band blocking specification (see Section 4.3 on page 25) shall apply. The interfering signal shall be Bluetooth-modulated (see section 4.8 on page 27). The BER shall be ≤ 0.1%. The signal to interference ratio shall be:

Requirement	Ratio
Co-Channel interference, $C/I_{co\text{-}channel}$	11 dB[1)]
Adjacent (1 MHz) interference, C/I_{1MHz}	0 dB[1]
Adjacent (2 MHz) interference, C/I_{2MHz}	-30 dB
Adjacent (≥3 MHz) interference, $C/I_{\geq 3MHz}$	-40 dB
Image frequency Interference[2) 3)], C/I_{Image}	-9 dB[1]
Adjacent (1 MHz) interference to in-band image frequency, $C/I_{Image\pm 1MHz}$	-20 dB[1]

Table 4.1: Interference performance

Note 1. These specifications are tentative and will be fixed within 18 months after the release of the Bluetooth specification version 1.0. Implementations have to fulfil the final specification after a 3-years' convergence period starting at the release of the Bluetooth specification version 1.0. During the convergence period, devices need to achieve a co-channel interference resistance of +14 dB, an ACI (@1MHz) resistance of +4 dB, Image frequency interference resistance of –6 dB and an ACI to in-band image frequency resistance of –16 dB.

Note 2. In-band image frequency

Note 3. If the image frequency ≠ n*1 MHz, than the image reference frequency is defined as the closest n*1 MHz frequency.

Note 4. If two adjacent channel specifications from Table 4.1 are applicable to the same channel, the more relaxed specification applies.

These specifications are only to be tested at nominal temperature conditions with a receiver hopping on one frequency, meaning that the synthesizer must change frequency between receive slot and transmit slot, but always return to the same receive frequency.

Frequencies where the requirements are not met are called spurious response frequencies. Five spurious response frequencies are allowed at frequencies with a distance of ≥2 MHz from the wanted signal. On these spurious response frequencies a relaxed interference requirement C/I = -17 dB shall be met.

4.3 OUT-OF-BAND BLOCKING

The Out of band blocking is measured with the wanted signal 3 dB over the reference sensitivity level. The interfering signal shall be a continuous wave signal. The BER shall be ≤ 0.1%. The Out of band blocking shall fulfil the following requirements:

Interfering Signal Frequency	Interfering Signal Power Level
30 MHz - 2000 MHz	-10 dBm
2000 - 2399 MHz	-27 dBm
2498 – 3000 MHz	-27 dBm
3000 MHz – 12.75 GHz	-10 dBm

Table 4.2: Out of Band blocking requirements

24 exceptions are permitted which are dependent upon the given receive channel frequency and are centered at a frequency which is an integer multiple of 1 MHz. At 19 of these spurious response frequencies a relaxed power level -50 dBm of the interferer may used to achieve a BER of 0.1%. At the remaining 5 spurious response frequencies the power level is arbitrary.

4.4 INTERMODULATION CHARACTERISTICS

The reference sensitivity performance, BER = 0.1%, shall be met under the following conditions.

- The wanted signal at frequency f_0 with a power level 6 dB over the reference sensitivity level.
- A static sine wave signal at f_1 with a power level of –39 dBm
- A Bluetooth modulated signal (see Section 4.8 on page 27) at f_2 with a power level of -39 dBm

Such that $f_0=2f_1-f_2$ and $|f_2-f_1|=n*1$ MHz, where n can be 3, 4, or 5. The system must fulfil one of the three alternatives.

4.5 MAXIMUM USABLE LEVEL

The maximum usable input level the receiver shall operate at shall be better than –20 dBm. The BER shall be less or equal to 0,1% at –20* dBm input power.

4.6 SPURIOUS EMISSIONS

The spurious emission for a Bluetooth receiver shall not be more than:

Frequency Band	Requirement
30 MHz - 1 GHz	-57 dBm
1 GHz – 12.75 GHz	-47 dBm

Table 4.3: Out-of-band spurious emission

The measured power should be measured in a 100 kHz bandwidth.

4.7 RECEIVER SIGNAL STRENGTH INDICATOR (OPTIONAL)

A transceiver that wishes to take part in a power-controlled link must be able to measure its own receiver signal strength and determine if the transmitter on the other side of the link should increase or decrease its output power level. A Receiver Signal Strength Indicator (RSSI) makes this possible.

The way the power control is specified is to have a golden receive power. This golden receive power is defined as a range with a low limit and a high limit. The RSSI must have a minimum dynamic range equal to this range. The RSSI must have an absolute accuracy of ±4dB or better when the receive signal power is –60 dBm. In addition, a minimum range of 20±6 dB must be covered, starting from –60 dB and up (see Figure 4.1 on page 26).

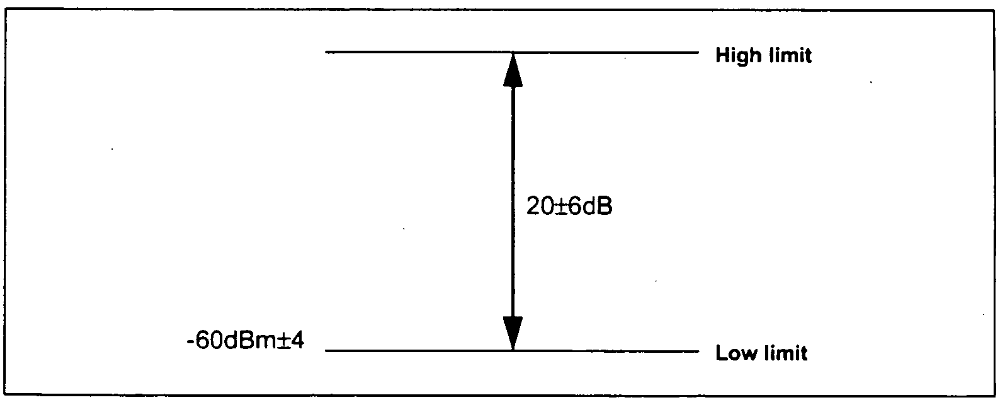


Figure 4.1: RSSI dynamic range and accuracy

4.8 REFERENCE INTERFERENCE-SIGNAL DEFINITION

A Bluetooth modulated interfering signal is defined as:

Modulation = GFSK

Modulation index = 0.32±1%

BT= 0.5±1%

Bit Rate = 1 Mbps ±1 ppm

Modulating Data = PRBS9

Frequency accuracy better than ±1 ppm.

5 APPENDIX A

5.1 NOMINAL TEST CONDITIONS (NTC)

5.1.1 Nominal temperature

The nominal temperature conditions for tests shall be +15 to +35 °C. When it is impractical to carry out the test under this condition a note to this effect, stating the ambient temperature, shall be recorded. The actual value during the test shall be recorded in the test report.

5.1.2 Nominal Power source

5.1.2.1 Mains Voltage

The nominal test voltage for equipment to be connected to the mains shall be the nominal mains voltage. The nominal voltage shall be declared voltage or any of the declared voltages for which the equipment was designed. The frequency of the test power source corresponding to the AC mains shall be within 2% of the nominal frequency.

5.1.2.2 Lead-acid battery power sources used in vehicles

When radio equipment is intended for operation from the alternator-fed lead-acid battery power sources which are standard in vehicles, then the nominal test voltage shall be 1.1 times the nominal voltage of the battery (6V, 12V, etc.).

5.1.2.3 Other power sources

For operation from other power sources or types of battery (primary or secondary), the nominal test voltage shall be as declared by the equipment manufacturer. This shall be recorded in the test report.

5.2 EXTREME TEST CONDITIONS

5.2.1 Extreme temperatures

The extreme temperature range is defined as the largest temperature range given by the combination of:

- · The minimum temperature range 0 °C to +35 °C
- · The product operating temperature range declared by the manufacturer.

This extreme temperature range and the declared operating temperature range shall be recorded in the test report.

5.2.2 Extreme power source voltages

Tests at extreme power source voltages specified below are not required when the equipment under test is designed for operation as part of and powered by another system or piece of equipment. Where this is the case, the limit values of the host system or host equipment shall apply. The appropriate limit values shall be declared by the manufacturer and recorded in the test report.

5.2.2.1 Mains voltage

The extreme test voltage for equipment to be connected to an AC mains source shall be the nominal mains voltage ±10%.

5.2.2.2 Lead-acid battery power source used on vehicles

When radio equipment is intended for operation from the alternator-fed lead-acid battery power sources which are standard in vehicles, then extreme test voltage shall be 1.3 and 0.9 times the nominal voltage of the battery (6V, 12V etc.)

5.2.2.3 Power sources using other types of batteries

The lower extreme test voltage for equipment with power sources using the following types of battery, shall be

a) for Leclanché, alkaline, or lithium type battery: 0.85 times the nominal voltage of the battery

b) for the mercury or nickel-cadmium types of battery: 0.9 times the nominal voltage of the battery.

In both cases, the upper extreme test voltage shall be 1.15 times the nominal voltage of the battery.

5.2.2.4 Other power sources

For equipment using other power sources, or capable of being operated from a variety of power sources (primary or secondary), the extreme test voltages shall be those declared by the manufacturer. These shall be recorded in the test report.

6 APPENDIX B

The Radio parameters shall be tested in the following conditions

Parameter	Temperature	Power source
Output Power	ETC	ETC
Power control	NTC	NTC
Modulation index	ETC	ETC
Initial Carrier Frequency accuracy	ETC	ETC
Carrier Frequency drift	ETC	ETC
In-band spurious emissions	ETC	ETC
Out-of-band Spurious Emissions	ETC	ETC
Sensitivity	ETC	ETC
Interference Performance	NTC	NTC
Intermodulation Characteristics	NTC	NTC
Out-of-band blocking	NTC	NTC
Maximum Usable Level	NTC	NTC
Receiver Signal Strength Indicator	NTC	NTC

ETC = Extreme Test Conditions
NTC = Nominal Test Conditions

Part B

BASEBAND SPECIFICATION

This document describes the specifications of the Bluetooth link controller which carries out the baseband protocols and other low-level link routines.

CONTENTS

1 GENERAL DESCRIPTION

Bluetooth is a short-range radio link intended to replace the cable(s) connecting portable and/or fixed electronic devices. Key features are robustness, low complexity, low power, and low cost.

Bluetooth operates in the unlicensed ISM band at 2.4 GHz. A frequency hop transceiver is applied to combat interference and fading. A shaped, binary FM modulation is applied to minimize transceiver complexity. The symbol rate is 1 Ms/s. A slotted channel is applied with a nominal slot length of 625 μs. For full duplex transmission, a Time-Division Duplex (TDD) scheme is used. On the channel, information is exchanged through packets. Each packet is transmitted on a different hop frequency. A packet nominally covers a single slot, but can be extended to cover up to five slots.

The Bluetooth protocol uses a combination of circuit and packet switching. Slots can be reserved for synchronous packets. Bluetooth can support an asynchronous data channel, up to three simultaneous synchronous voice channels, or a channel which simultaneously supports asynchronous data and synchronous voice. Each voice channel supports a 64 kb/s synchronous (voice) channel in each direction. The asynchronous channel can support maximal 723.2 kb/s asymmetric (and still up to 57.6 kb/s in the return direction), or 433.9 kb/s symmetric.

The Bluetooth system consists of a radio unit (see Radio Specification), a link control unit, and a support unit for link management and host terminal interface functions, see Figure 1.1 on page 41. The current document describes the specifications of the Bluetooth link controller, which carries out the baseband protocols and other low-level link routines. Link layer messages for link set-up and control are defined in the Link Manager Protocol on page 185.

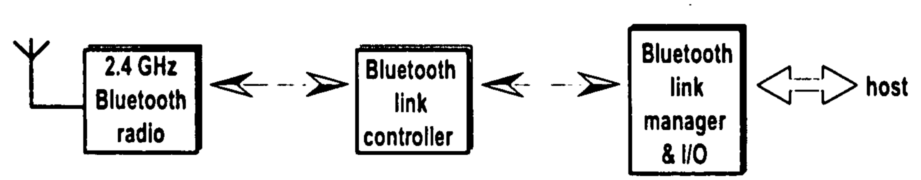


Figure 1.1: Different functional blocks in the Bluetooth system

The Bluetooth system provides a point-to-point connection (only two Bluetooth units involved), or a point-to-multipoint connection, see Figure 1.2 on page 42. In the point-to-multipoint connection, the channel is shared among several Bluetooth units. Two or more units sharing the same channel form a ***piconet***. One Bluetooth unit acts as the master of the piconet, whereas the other unit(s)

acts as slave(s). Up to seven slaves can be active in the piconet. In addition, many more slaves can remain locked to the master in a so-called parked state. These parked slaves cannot be active on the channel, but remain synchronized to the master. Both for active and parked slaves, the channel access is controlled by the master.

Multiple piconets with overlapping coverage areas form a ***scatternet***. Each piconet can only have a single master. However, slaves can participate in different piconets on a time-division multiplex basis. In addition, a master in one piconet can be a slave in another piconet. The piconets shall not be time- or frequency-synchronized. Each piconet has its own hopping channel.

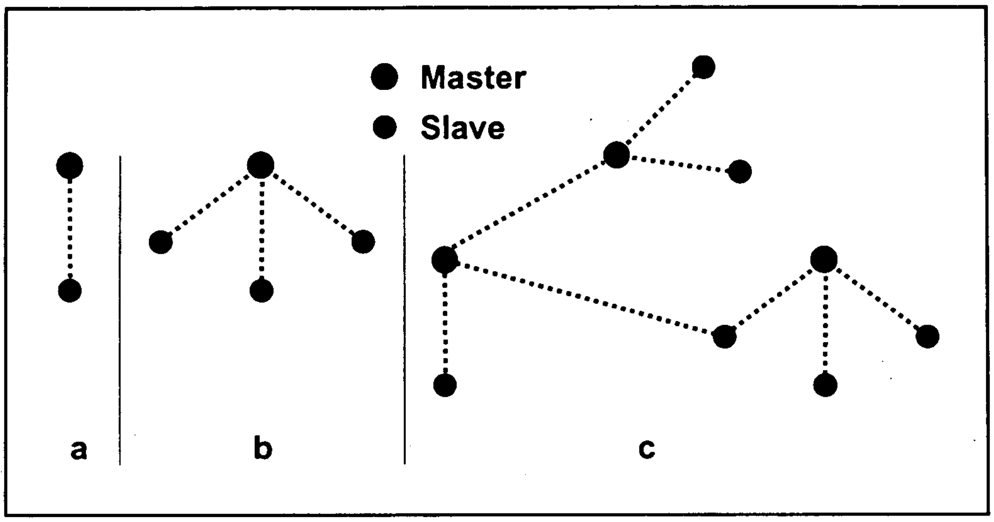


Figure 1.2: Piconets with a single slave operation (a), a multi-slave operation (b) and a scatternet operation (c).

2 PHYSICAL CHANNEL

2.1 FREQUENCY BAND AND RF CHANNELS

Bluetooth operates in the 2.4 GHz ISM band. Although globally available, the exact location and the width of the band may differ by country. In the US and Europe, a band of 83.5 MHz width is available; in this band, 79 RF channels spaced 1 MHz apart are defined. In Japan, Spain, and France, a smaller band is available; in this band, 23 RF channels spaced 1 MHz apart are defined.

Country	Frequency Range	RF Channels	
Europe[*] & USA	2400 - 2483.5 MHz	f = 2402 + k MHz	k= 0,...,78
Japan	2471 - 2497 MHz	f = 2473 + k MHz	k= 0,...,22
Spain	2445 - 2475 MHz	f = 2449 + k MHz	k= 0,...,22
France	2446.5 - 2483.5 MHz	f = 2454 + k MHz	k= 0,...,22

Table 2.1: Available RF channels

*. except Spain and France

2.2 CHANNEL DEFINITION

The channel is represented by a pseudo-random hopping sequence hopping through the 79 or 23 RF channels. The hopping sequence is unique for the piconet and is determined by the Bluetooth device address of the master; the phase in the hopping sequence is determined by the Bluetooth clock of the master. The channel is divided into time slots where each slot corresponds to an RF hop frequency. Consecutive hops correspond to different RF hop frequencies. The nominal hop rate is 1600 hops/s. All Bluetooth units participating in the piconet are time- and hop-synchronized to the channel.

2.3 TIME SLOTS

The channel is divided into time slots, each 625 μs in length. The time slots are numbered according to the Bluetooth clock of the piconet master. The slot numbering ranges from 0 to 2^{27}-1 and is cyclic with a cycle length of 2^{27}.

In the time slots, master and slave can transmit packets.

A TDD scheme is used where master and slave alternatively transmit, see Figure 2.1 on page 44. The master shall start its transmission in even-numbered time slots only, and the slave shall start its transmission in odd-numbered time slots only. The packet start shall be aligned with the slot start. Packets transmitted by the master or the slave may extend over up to five time slots.

The RF hop frequency shall remain fixed for the duration of the packet. For a single packet, the RF hop frequency to be used is derived from the current Bluetooth clock value. For a multi-slot packet, the RF hop frequency to be used for the entire packet is derived from the Bluetooth clock value in the first slot of the packet. The RF hop frequency in the first slot after a multi-slot packet shall use the frequency as determined by the current Bluetooth clock value. Figure 2.2 on page 44 illustrates the hop definition on single- and multi-slot packets. If a packet occupies more than one time slot, the hop frequency applied shall be the hop frequency as applied in the time slot where the packet transmission was started.

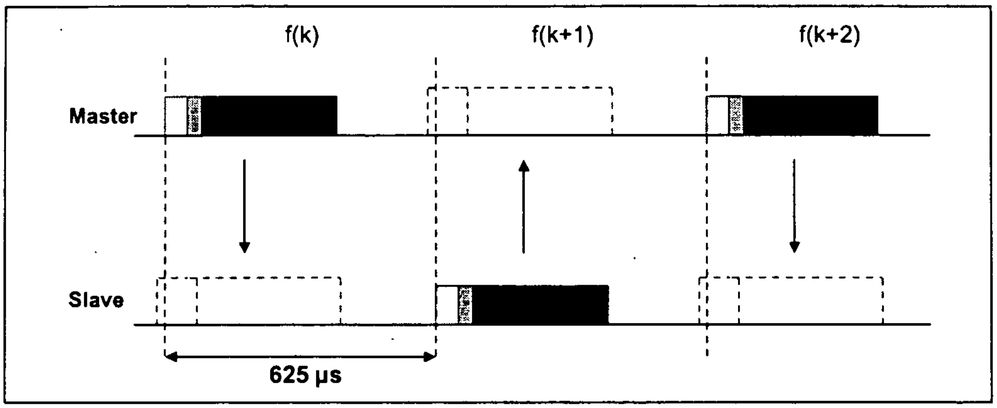


Figure 2.1: TDD and timing

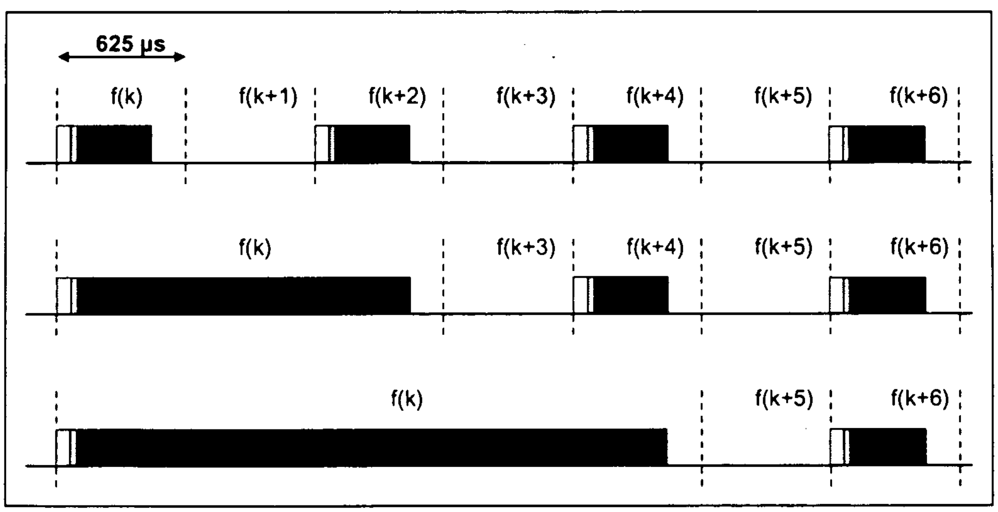


Figure 2.2: Multi-slot packets

2.4 MODULATION AND BIT RATE

The data transmitted has a symbol rate of 1 Ms/s. A Gaussian-shaped, binary FSK modulation is applied with a *BT* product of 0.5. A binary one is represented by a positive frequency deviation, a binary zero by a negative frequency deviation. The maximum frequency deviation shall be between 140 kHz and 175 kHz.

3 PHYSICAL LINKS

3.1 GENERAL

Between master and slave(s), different types of links can be established. Two link types have been defined:

- Synchronous Connection-Oriented (SCO) link
- Asynchronous Connection-Less (ACL) link

The SCO link is a point-to-point link between a master and a single slave in the piconet. The master maintains the SCO link by using reserved slots at regular intervals. The ACL link is a point-to-multipoint link between the master and all the slaves participating on the piconet. In the slots not reserved for the SCO link(s), the master can establish an ACL link on a per-slot basis to any slave, including the slave(s) already engaged in an SCO link.

3.2 SCO LINK

The SCO link is a symmetric, point-to-point link between the master and a specific slave. The SCO link reserves slots and can therefore be considered as a circuit-switched connection between the master and the slave. The SCO link typically supports time-bounded information like voice. The master can support up to three SCO links to the same slave or to different slaves. A slave can support up to three SCO links from the same master, or two SCO links if the links originate from different masters. SCO packets are never retransmitted.

The master will send SCO packets at regular intervals, the so-called SCO interval T_{SCO} (counted in slots) to the slave in the reserved master-to-slave slots. The SCO slave is always allowed to respond with an SCO packet in the following slave-to-master slot unless a different slave was addressed in the previous master-to-slave slot. If the SCO slave fails to decode the slave address in the packet header, it is still allowed to return an SCO packet in the reserved SCO slot.

The SCO link is established by the master sending an SCO setup message via the LM protocol. This message will contain timing parameters such as the SCO interval T_{SCO} and the offset D_{SCO} to specify the reserved slots.

In order to prevent clock wrap-around problems, an initialization flag in the LMP setup message indicates whether initialization procedure 1 or 2 is being used. The slave shall apply the initialization method as indicated by the initialization flag. The master uses initialization 1 when the MSB of the current master clock (CLK_{27}) is 0; it uses initialization 2 when the MSB of the current master clock (CLK_{27}) is 1. The master-to-slave SCO slots reserved by the master and the slave shall be initialized on the slots for which the clock satisfies the following equation:

$CLK_{27\text{-}1} \bmod T_{SCO} = D_{SCO}$ for initialization 1

$(\overline{CLK}_{27}, CLK_{26\text{-}1}) \bmod T_{SCO} = D_{SCO}$ for initialization 2

The slave-to-master SCO slots shall directly follow the reserved master-to-slave SCO slots. After initialization, the clock value CLK(k+1) for the next master-to-slave SCO slot is found by adding the fixed interval T_{SCO} to the clock value of the current master-to-slave SCO slot:

$$CLK(k+1) = CLK(k) + T_{SCO}$$

3.3 ACL LINK

In the slots not reserved for SCO links, the master can exchange packets with any slave on a per-slot basis. The ACL link provides a packet-switched connection between the master and all active slaves participating in the piconet. Both asynchronous and isochronous services are supported. Between a master and a slave only a single ACL link can exist. For most ACL packets, packet retransmission is applied to assure data integrity.

A slave is permitted to return an ACL packet in the slave-to-master slot if and only if it has been addressed in the preceding master-to-slave slot. If the slave fails to decode the slave address in the packet header, it is not allowed to transmit.

ACL packets not addressed to a specific slave are considered as broadcast packets and are read by every slave. If there is no data to be sent on the ACL link and no polling is required, no transmission shall take place.

4 PACKETS

4.1 GENERAL FORMAT

The bit ordering when defining packets and messages in the *Baseband Specification*, follows the *Little Endian format*, i.e., the following rules apply:

- The *least significant bit* (LSB) corresponds to b_0;
- The LSB is the first bit sent over the air;
- In illustrations, the LSB is shown on the left side;

The baseband controller interprets the first bit arriving from a higher software layer as b_0; i.e. this is the first bit to be sent over the air. Furthermore, data fields generated internally at baseband level, such as the packet header fields and payload header length, are transmitted with the LSB first. For instance, a 3-bit parameter X=3 is sent as $b_0 b_1 b_2 = 110$ over the air where 1 is sent first and 0 is sent last.

The data on the piconet channel is conveyed in packets. The general packet format is shown in Figure 4.1 on page 47. Each packet consists of 3 entities: the access code, the header, and the payload. In the figure, the number of bits per entity is indicated.

LSB 72	54	0 - 2745 MSB
ACCESS CODE	HEADER	PAYLOAD

Figure 4.1: Standard packet format.

The access code and header are of fixed size: 72 bits and 54 bits respectively. The payload can range from zero to a maximum of 2745 bits. Different packet types have been defined. Packets may consist of the (shortened) access code only (see ID packet on page 55), of the access code – header, or of the access code – header – payload.

4.2 ACCESS CODE

Each packet starts with an access code. If a packet header follows, the access code is 72 bits long, otherwise the access code is 68 bits long. This access code is used for synchronization, DC offset compensation and identification. The access code identifies all packets exchanged on the channel of the piconet: all packets sent in the same piconet are preceded by the same channel access code. In the receiver of the Bluetooth unit, a sliding correlator correlates against the access code and triggers when a threshold is exceeded. This trigger signal is used to determine the receive timing.

The access code is also used in paging and inquiry procedures. In this case, the access code itself is used as a signalling message and neither a header nor a payload is present.

The access code consists of a preamble, a sync word, and possibly a trailer, see Figure 4.2 on page 48. For details see Section 4.2.1 on page 48.

LSB 4	64	4 MSB
PREAMBLE	SYNC WORD	TRAILER

Figure 4.2: Access code format

4.2.1 Access code types

There are three different types of access codes defined:

- Channel Access Code (CAC)
- Device Access Code (DAC)
- Inquiry Access Code (IAC)

The respective access code types are used for a Bluetooth unit in different operating modes. The channel access code identifies a piconet. This code is included in all packets exchanged on the piconet channel. The device access code is used for special signalling procedures, e.g., paging and response to paging. For the inquiry access code there are two variations. A general inquiry access code (GIAC) is common to all devices. The GIAC can be used to discover which other Bluetooth units are in range. The dedicated inquiry access code (DIAC) is common for a dedicated group of Bluetooth units that share a common characteristic. The DIAC can be used to discover only these dedicated Bluetooth units in range.

The CAC consists of a **preamble**, **sync word**, and **trailer** and its total length is 72 bits. When used as self-contained messages without a header, the DAC and IAC do not include the trailer bits and are of length 68 bits.

The different access code types use different Lower Address Parts (LAPs) to construct the sync word. The LAP field of the BD address is explained in Section 13.1 on page 143. A summary of the different access code types can be found in Table 4.1 on page 49.

Code type	LAP	Code length	Comments
CAC	Master	72	See also Section 13.2 on page 143
DAC	Paged unit	68/72*	
GIAC	Reserved	68/72*	
DIAC	Dedicated	68/72*	

Table 4.1: Summary of access code types.

*. length 72 is only used in combination with FHS packets

4.2.2 Preamble

The preamble is a fixed zero-one pattern of 4 symbols used to facilitate DC compensation. The sequence is either 1010 or 0101, depending whether the LSB of the following sync word is 1 or 0, respectively. The preamble is shown in Figure 4.3 on page 49.

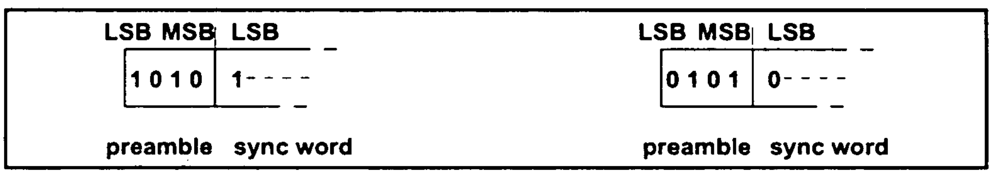


Figure 4.3: Preamble

4.2.3 Sync Word

The sync word is a 64-bit code word derived from a 24 bit address (LAP); for the CAC the master's LAP is used; for the GIAC and the DIAC, reserved, dedicated LAPs are used; for the DAC, the slave unit LAP is used. The construction guarantees large Hamming distance between sync words based on different LAPs. In addition, the good autocorrelation properties of the sync word improve on the timing synchronization process. The derivation of the sync word is described in Section 13.2 on page 143

4.2.4 Trailer

The trailer is appended to the sync word as soon as the packet header follows the access code. This is typically the case with the CAC, but the trailer is also used in the DAC and IAC when these codes are used in FHS packets exchanged during page response and inquiry response procedures.

The trailer is a fixed zero-one pattern of four symbols. The trailer together with the three MSBs of the syncword form a 7-bit pattern of alternating ones and zeroes which may be used for extended DC compensation. The trailer sequence is either 1010 or 0101 depending on whether the MSB of the sync word is 0 or 1, respectively. The choice of trailer is illustrated in Figure 4.4 on page 50.

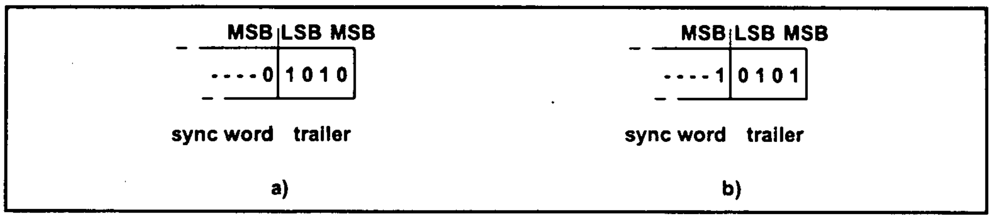


Figure 4.4: Trailer in CAC when MSB of sync word is 0 (a), and when MSB of sync word is 1 (b).

4.3 PACKET HEADER

The header contains link control (LC) information and consists of 6 fields:

- AM_ADDR: 3- bit active member address
- TYPE: 4-bit type code
- FLOW: 1-bit flow control
- ARQN: 1-bit acknowledge indication
- SEQN: 1-bit sequence number
- HEC: 8-bit header error check

The total header, including the HEC, consists of 18 bits, see Figure 4.5 on page 51, and is encoded with a rate 1/3 FEC (not shown but described in Section 5.1 on page 67) resulting in a 54-bit header. Note that the AM_ADDR and TYPE fields are sent with their LSB first. The function of the different fields will be explained next.

LSB 3	4	1	1	1	8 MSB
AM_ADDR	TYPE	FLOW	ARQN	SEQN	HEC

Figure 4.5: Header format.

4.3.1 AM_ADDR

The AM_ADDR represents a member address and is used to distinguish between the active members participating on the piconet. In a piconet, one or more slaves are connected to a single master. To identify each slave separately, each slave is assigned a temporary 3-bit address to be used when it is active. Packets exchanged between the master and the slave all carry the AM_ADDR of this slave; that is, the AM_ADDR of the slave is used in both master-to-slave packets and in the slave-to-master packets. The all-zero address is reserved for broadcasting packets from the master to the slaves. An exception is the FHS packet which may use the all-zero member address but is *not* a broadcast message (Section 4.4.1.4 on page 56). Slaves that are disconnected or parked give up their AM_ADDR. A new AM_ADDR has to be assigned when they re-enter the piconet.

4.3.2 TYPE

Sixteen different types of packets can be distinguished. The 4-bit TYPE code specifies which packet type is used. Important to note is that the interpretation of the TYPE code depends on the physical link type associated with the packet. First, it shall be determined whether the packet is sent on an SCO link or an ACL link. Then it can be determined which type of SCO packet or ACL packet has been received. The TYPE code also reveals how many slots the current packet will occupy. This allows the non-addressed receivers to refrain

from listening to the channel for the duration of the remaining slots. In Section 4.4 on page 54, each packet type will be described in more detail.

4.3.3 FLOW

This bit is used for flow control of packets over the ACL link. When the RX buffer for the ACL link in the recipient is full and is not emptied, a STOP indication (FLOW=0) is returned to stop the transmission of data temporarily. Note, that the STOP signal only concerns ACL packets. Packets including only link control information (ID, POLL and NULL packets) or SCO packets can still be received. When the RX buffer is empty, a GO indication (FLOW=1) is returned. When no packet is received, or the received header is in error, a GO is assumed implicitly.

4.3.4 ARQN

The 1-bit acknowledgment indication ARQN is used to inform the source of a successful transfer of payload data with CRC, and can be positive acknowledge ACK or negative acknowledge NAK. If the reception was successful, an ACK (ARQN=1) is returned, otherwise a NAK (ARQN=0) is returned. When no return message regarding acknowledge is received, a NAK is assumed implicitly. NAK is also the default return information.

The ARQN is piggy-backed in the header of the return packet. The success of the reception is checked by means of a cyclic redundancy check (CRC) code. An unnumbered ARQ scheme which means that the ARQN relates to the latest received packet from the same source, is used. See Section 5.3 on page 68 for initialization and usage of this bit.

4.3.5 SEQN

The SEQN bit provides a sequential numbering scheme to order the data packet stream. For each new transmitted packet that contains data with CRC, the SEQN bit is inverted. This is required to filter out retransmissions at the destination; if a retransmission occurs due to a failing ACK, the destination receives the same packet twice. By comparing the SEQN of consecutive packets, correctly received retransmissions can be discarded. The SEQN has to be added due to a lack of packet numbering in the unnumbered ARQ scheme. See section 5.3.2 on page 70 for initialization and usage of the SEQN bit. For broadcast packets, a modified sequencing method is used, see Section 5.3.5 on page 72.

4.3.6 HEC

Each header has a header-error-check to check the header integrity. The HEC consists of an 8-bit word generated by the polynomial 647 (octal representation). Before generating the HEC, the HEC generator is initialized with an 8-bit value. For FHS packets sent in **master page respons** state, the slave upper

address part (UAP) is used. For FHS packets sent in **inquiry r spons** , the default check initialization (DCI, see Section 5.4) is used. In all other cases, the UAP of the master device is used. For the definition of Bluetooth device addresses, see Section 13.1 on page 143.

After the initialization, a HEC is calculated for the 10 header bits. Before checking the HEC, the receiver must initialize the HEC check circuitry with the proper 8-bit UAP (or DCI). If the HEC does not check, the entire packet is disregarded. More information can be found in Section 5.4 on page 73.

4.4 PACKET TYPES

The packets used on the piconet are related to the physical links they are used in. Up to now, two physical links are defined: the SCO link and the ACL link. For each of these links, 12 different packet types can be defined. Four control packets will be common to all link types: their TYPE code is unique irrespective of the link type.

To indicate the different packets on a link, the 4-bit TYPE code is used. The packet types have been divided into four segments. The first segment is reserved for the four control packets common to all physical link types; all four packet types have been defined. The second segment is reserved for packets occupying a single time slot; six packet types have been defined. The third segment is reserved for packets occupying three time slots; two packet types have been defined. The fourth segment is reserved for packets occupying five time slots; two packet types have been defined. The slot occupancy is reflected in the segmentation and can directly be derived from the type code. Table 4.2 on page 54 summarizes the packets defined so far for the SCO and ACL link types.

Segment	TYPE code $b_3b_2b_1b_0$	Slot occupancy	SCO link	ACL link
1	0000	1	NULL	NULL
	0001	1	POLL	POLL
	0010	1	FHS	FHS
	0011	1	DM1	DM1
2	0100	1	undefined	DH1
	0101	1	HV1	undefined
	0110	1	HV2	undefined
	0111	1	HV3	undefined
	1000	1	DV	undefined
	1001	1	undefined	AUX1
3	1010	3	undefined	DM3
	1011	3	undefined	DH3
	1100	3	undefined	undefined
	1101	3	undefined	undefined
4	1110	5	undefined	DM5
	1111	5	undefined	DH5

Table 4.2: Packets defined for SCO and ACL link types

4.4.1 Common packet types

There are five common packets. In addition to the types listed in segment 1 of the previous table, there is the ID packet not listed. Each packet will now be examined in more detail.

4.4.1.1 ID packet

The identity or ID packet consists of the device access code (DAC) or inquiry access code (IAC). It has a fixed length of 68 bits. It is a very robust packet since the receiver uses a bit correlator to match the received packet to the known bit sequence of the ID packet. The packet is used, for example, in paging, inquiry, and response routines.

4.4.1.2 NULL packet

The NULL packet has no payload and therefore consists of the channel access code and packet header only. Its total (fixed) length is 126 bits. The NULL packet is used to return link information to the source regarding the success of the previous transmission (ARQN), or the status of the RX buffer (FLOW). The NULL packet itself does not have to be acknowledged.

4.4.1.3 POLL packet

The POLL packet is very similar to the NULL packet. It does not have a payload either. In contrast to the NULL packet, it requires a confirmation from the recipient. It is not a part of the ARQ scheme. The POLL packet does not affect the ARQN and SEQN fields. Upon reception of a POLL packet the slave must respond with a packet. This return packet is an implicit acknowledgement of the POLL packet. This packet can be used by the master in a piconet to poll the slaves, which must then respond even if they do not have information to send.

4.4.1.4 FHS packet

The FHS packet is a special control packet revealing, among other things, the Bluetooth device address and the clock of the sender. The payload contains 144 information bits plus a 16-bit CRC code. The payload is coded with a rate 2/3 FEC which brings the gross payload length to 240 bits. The FHS packet covers a single time slot.

Figure 4.6 on page 56 illustrates the format and contents of the FHS payload. The payload consists of eleven fields. The FHS packet is used in page master response, inquiry response and in master slave switch. In page master response or master slave switch, it is retransmitted until its reception is acknowledged or a timeout has exceeded. In inquiry response, the FHS packet is not acknowledged. The FHS packet contains real-time clock information. This clock information is updated before each retransmission. The retransmission of the FHS payload is thus somewhat different from the retransmission of ordinary data payloads where the same payload is used for each retransmission. The FHS packet is used for frequency hop synchronization before the piconet channel has been established, or when an existing piconet changes to a new piconet. In the former case, the recipient has not been assigned an active member address yet, in which case the AM_ADDR field in the FHS packet header is set to all-zeroes; however, the FHS packet should not be considered as a broadcast packet. In the latter case the slave already has an AM_ADDR in the existing piconet, which is then used in the FHS packet header.

LSB ... MSB

34	24	2	2	2	8	16	24	3	26	3
Parity bits	LAP	Un-defined	SR	SP	UAP	NAP	Class of device	AM_ADDR	CLK_{27-2}	Page scan mode

Figure 4.6: Format of the FHS payload

Each field is described in more detail below:

Parity bits	This 34-bit field contains the parity bits that form the first part of the sync word of the access code of the unit that sends the FHS packet. These bits are derived from the LAP as described in section 13.2 on page 143.
LAP	This 24-bit field contains the lower address part of the unit that sends the FHS packet.
Undefined	This 2-bit field is reserved for future use and shall be set to zero.
SR	This 2-bit field is the scan repetition field and indicates the interval between two consecutive page scan windows, see also Table 4.4 and Table 10.1 on page 101

Table 4.3: Description of the FHS payload

SP	This 2-bit field is the scan period field and indicates the period in which the mandatory page scan mode is applied after transmission of an inquiry response message, see also Table 4.5 and Table 10.6 on page 112.
UAP	This 8-bit field contains the upper address part of the unit that sends the FHS packet.
NAP	This 16-bit field contains the non-significant address part of the unit that sends the FHS packet (see also section 13.1 on page 143 for LAP, UAP, and NAP).
Class of device	This 24-bit field contains the class of device of the unit that sends the FHS packet. The class of device has not been defined yet.
AM_ADDR	This 3-bit field contains the member address the recipient shall use if the FHS packet is used at call setup or master-slave switch. A slave responding to a master or a unit responding to an inquiry request message shall include an all-zero AM_ADDR field if it sends the FHS packet.
$CLK_{27\text{-}2}$	**This 26-bit field contains the value of the native system clock of the unit that sends the FHS packet, sampled at the beginning of the transmission of the access code of this FHS packet. This clock value has a resolution of 1.25ms (two-slot interval). For each new transmission, this field is updated so that it accurately reflects the real-time clock value.**
Page scan mode	This 3-bit field indicates which scan mode is used by default by the sender of the FHS packet. The interpretation of the page scan mode is illustrated in Table 4.6. Currently, the standard supports one mandatory scan mode and up to three optional scan modes (see also "Appendix VII" on page 999).

Table 4.3: Description of the FHS payload

SR bit format b_1b_0	**SR mode**
00	R0
01	R1
10	R2
11	reserved

Table 4.4: Contents of SR field

SP bit format b_1b_0	**SP mode**
00	P0
01	P1
10	P2
11	reserved

Table 4.5: Contents of SP field

Bit format $b_2b_1b_0$	Page scan mode
000	Mandatory scan mode
001	Optional scan mode I
010	Optional scan mode II
011	Optional scan mode III
100	Reserved for future use
101	Reserved for future use
110	Reserved for future use
111	Reserved for future use

Table 4.6: Contents of page scan mode field

The LAP, UAP, and NAP together form the 48-bit IEEE address of the unit that sends the FHS packet. Using the parity bits and the LAP, the recipient can directly construct the channel access code of the sender of the FHS packet.

4.4.1.5 DM1 packet

DM1 serves as part of segment 1 in order to support control messages in any link type. However, it can also carry regular user data. Since the DM1 packet is recognized on the SCO link, it can interrupt the synchronous information to send control information. Since the DM1 packet can be regarded as an ACL packet, it will be discussed in Section 4.4.3 on page 60.

4.4.2 SCO packets

SCO packets are used on the synchronous SCO link. The packets do not include a CRC and are never retransmitted. SCO packets are routed to the synchronous I/O (voice) port. Up to now, three pure SCO packets have been defined. In addition, an SCO packet is defined which carries an asynchronous data field in addition to a synchronous (voice) field. The SCO packets defined so far are typically used for 64 kb/s speech transmission.

4.4.2.1 HV1 packet

The **HV1** packet carries 10 information bytes. The bytes are protected with a rate 1/3 FEC. No CRC is present. The payload length is fixed at 240 bits. There is no payload header present.

HV packets are typically used for voice transmission. HV stands for High-quality Voice. The voice packets are never retransmitted and need no CRC.

An HV1 packet can carry 1.25ms of speech at a 64 kb/s rate. In that case, an HV1 packet has to be sent every two time slots (T_{SCO}=2).

4.4.2.2 HV2 packet

The **HV2** packet carries 20 information bytes. The bytes are protected with a rate 2/3 FEC. No CRC is present. The payload length is fixed at 240 bits. There is no payload header present.

If the HV2 packet is used for voice at a 64 kb/s rate, it can carry 2.5ms of speech. In that case, an HV2 packet has to be sent every four time slots (T_{SCO}=4).

4.4.2.3 HV3 packet

The **HV3** packet carries 30 information bytes. The bytes are not protected by FEC. No CRC is present. The payload length is fixed at 240 bits. There is no payload header present.

If the HV3 packet is used for voice at a 64 kb/s rate, it can carry 3.75ms of speech. In that case, an HV3 packet has to be sent every six time slots (T_{SCO}=6).

4.4.2.4 DV packet

The **DV** packet is a combined data - voice packet. The payload is divided into a voice field of 80 bits and a data field containing up to 150 bits, see Figure 4.7. The voice field is not protected by FEC. The data field contains up to 10 information bytes (including the 1-byte payload header) and includes a 16-bit CRC. The data field is encoded with a rate 2/3 FEC. If necessary, extra zeroes are appended to assure that the total number of payload bits is a multiple of 10 prior to FEC encoding. Since the **DV** packet has to be sent at regular intervals due to its synchronous (voice) contents, it is listed under the SCO packet types. The voice and data fields are treated completely separate. The voice field is handled like normal SCO data and is never retransmitted; that is, the voice field is always new. The data field is checked for errors and is retransmitted if necessary.

LSB 72	54	80	32 - 150 MSB
ACCESS CODE	HEADER	VOICE FIELD	DATA FIELD

Figure 4.7: DV packet format

4.4.3 ACL packets

ACL packets are used on the asynchronous links. The information carried can be user data or control data. Including the DM1 packet, seven ACL packets have been defined. Six of the ACL packets contain a CRC code and retransmission is applied if no acknowledgement of proper reception is received (except in case a flush operation is carried out, see Section 5.3.3 on page 71). The 7th ACL packet, the AUX1 packet, has no CRC and is not retransmitted.

4.4.3.1 DM1 packet

The DM1 packet is a packet that carries data information only. DM stands for Data – Medium rate. The payload contains up to 18 information bytes (including the 1-byte payload header) plus a 16-bit CRC code. The DM1 packet may cover up to a single time slot. The information plus CRC bits are coded with a rate 2/3 FEC which adds 5 parity bits to every 10-bit segment. If necessary, extra zeros are appended after the CRC bits to get the total number of bits (information bits, CRC bits, and tail bits) equal a multiple of 10. The payload header in the DM1 packet is only 1 byte long, see Figure 4.8 on page 62. The length indicator in the payload header specifies the number of user bytes (excluding payload header and the CRC code).

4.4.3.2 DH1 packet

This packet is similar to the DM1 packet, except that the information in the payload is not FEC encoded. As a result, the DH1 packet can carry up to 28 information bytes plus a 16-bit CRC code. DH stands for Data – High rate. The DH1 packet may cover up to a single time slot.

4.4.3.3 DM3 packet

The DM3 packet is a DM1 packet with an extended payload. The DM3 packet may cover up to three time slots. The payload contains up to 123 information bytes (including the 2-bytes payload header) plus a 16-bit CRC code. The payload header in the DM3 packet is 2 bytes long, see Figure 4.9 on page 62. The length indicator in the payload header specifies the number of user bytes (excluding payload header and the CRC code). When a DM3 packet is sent or received, the RF hop frequency shall not change for a duration of three time slots (the first time slot being the slot where the channel access code was transmitted).

4.4.3.4 DH3 packet

This packet is similar to the DM3 packet, except that the information in the payload is not FEC encoded. As a result, the DH3 packet can carry up to 185 information bytes (including the two bytes payload header) plus a 16-bit CRC code.

The DH3 packet may cover three time slots. When a DH3 packet is sent or received, the hop frequency shall not change for a duration of three time slots (the first time slot being the slot where the channel access code was transmitted).

4.4.3.5 DM5 packet

The DM5 packet is a DM1 packet with an extended payload. The DM5 packet may cover up to five time slots. The payload contains up to 226 information bytes (including the 2-bytes payload header) plus a 16-bit CRC code. The payload header in the DM5 packet is 2 bytes long. The length indicator in the payload header specifies the number of user bytes (excluding payload header and the CRC code). When a DM5 packet is sent or received, the hop frequency shall not change for a duration of five time slots (the first time slot being the slot where the channel access code was transmitted).

4.4.3.6 DH5 packet

This packet is similar to the DM5 packet, except that the information in the payload is not FEC encoded. As a result, the DH5 packet can carry up to 341 information bytes (including the two bytes payload header) plus a 16-bit CRC code. The DH5 packet may cover five time slots. When a DH5 packet is sent or received, the hop frequency shall not change for a duration of five time slots (the first time slot being the slot where the channel access code was transmitted).

4.4.3.7 AUX1 packet

This packet resembles a DH1 packet but has no CRC code. The AUX1 packet can carry up to 30 information bytes (including the 1-byte payload header). The AUX1 packet may cover up to a single time slot.

4.5 PAYLOAD FORMAT

In the previous packet overview, several payload formats were considered. In the payload, two fields are distinguished: the (synchronous) voice field and the (asynchronous) data field. The ACL packets only have the data field and the SCO packets only have the voice field – with the exception of the DV packets which have both.

4.5.1 Voice field

The voice field has a fixed length. For the HV packets, the voice field length is 240 bits; for the DV packet the voice field length is 80 bits. No payload header is present.

4.5.2 Data field

The data field consists of three segments: a payload header, a payload body, and possibly a CRC code (only the AUX1 packet does not carry a CRC code).

1. **Payload header**

   Only data fields have a payload header. The payload header is one or two bytes long. Packets in segments one and two have a 1-byte payload header; packets in segments three and four have a 2-bytes payload header. The payload header specifies the logical channel (2-bit L_CH indication), controls the flow on the logical channels (1-bit FLOW indication), and has a payload length indicator (5 bits and 9 bits for 1-byte and 2-bytes payload header, respectively). In the case of a 2-byte payload header, the length indicator is extended by four bits into the next byte. The remaining four bits of the second byte are reserved for future use and shall be set to zero. The formats of the 1-byte and 2-bytes payload headers are shown in Figure 4.8 on page 62 and Figure 4.9 on page 62.

LSB → MSB

2	1	5
L_CH	FLOW	LENGTH

Figure 4.8: Payload header format for single-slot packets.

LSB → MSB

2	1	9	4
L_CH	FLOW	LENGTH	UN-DEFINED

Figure 4.9: Payload header format for multi-slot packets.

The L_CH field is transmitted first, the length field last. In Table 4.7 on page 63, more details about the contents of the L_CH field are listed.

L_CH code b_1b_0	Logical Channel	Information
00	NA	undefined
01	UA/UI	Continuation fragment of an L2CAP message
10	UA/UI	Start of an L2CAP message or no fragmentation
11	LM	LMP message

Table 4.7: Logical channel L_CH field contents

An L2CAP message can be fragmented into several packets. Code 10 is used for an L2CAP packet carrying the first fragment of such a message; code 01 is used for continuing fragments. If there is no fragmentation, code 10 is used for every packet. Code 11 is used for LMP messages. Code 00 is reserved for future use.

The flow indicator in the payload is used to control the flow at the L2CAP level. It is used to control the flow per logical channel (when applicable). FLOW=1 means flow-on ("OK to send") and FLOW=0 means flow-off ("stop"). There are no strict real-time requirements on the flow bit in the payload header. Flow bit in the last correctly received payload header determines flow status. The link manager is responsible for setting and processing the flow bit in the payload header. Real-time flow control is carried out at the packet level by the link controller via the flow bit in the packet header (see Section 4.3.3 on page 52). With the payload flow bit, traffic from the remote end can be controlled. It is allowed to generate and send an ACL packet with payload load length zero. L2CAP start- and continue-fragment indications (L_CH=10 and L_CH=01) also retain their meaning when the payload length is equal to zero (i.e. an empty start-fragment should not be sent in the middle of an on-going L2CAP packet transmission). It is always safe to send an ACL packet with payload length=0 and L_CH=10. The payload flow bit has its own meaning for each logical channel (UA/I or LM), see Table 4.8 on page 63. On the LM channel, no flow control is applied and the payload flow bit is always set at one.

L_CH code b_1b_0	Usage and semantics of the ACL payload header FLOW bit
00	Not defined, reserved for future use.
01 or 10	Flow control of the UA/I channels (which are used to send L2CAP messages)
11	Always set FLOW=1 on transmission and ignore the bit on reception

Table 4.8: Use of payload header flow bit on the logical channels.

The length indicator indicates the number of bytes (i.e. 8-bit words) in the payload excluding the payload header and the CRC code; i.e. the payload body only. With reference to Figure 4.8 and Figure 4.9, the MSB of the length field in a 1-byte header is the last (right-most) bit in the payload

header; the MSB of the length field in a 2-byte header is the fourth bit (from left) of the second byte in the payload header.

2. **Payload body**

   The payload body includes the user host information and determines the effective user throughput. The length of the payload body is indicated in the length field of the payload header.

3. **CRC code generation**

   The 16-bit cyclic redundancy check code in the payload is generated by the CRC-CCITT polynomial `210041` (octal representation). It is generated in a way similar to the HEC. Before determining the CRC code, an 8-bit value is used to initialize the CRC generator. For the CRC code in the FHS packets sent in **master page response** state, the UAP of the slave is used. For the FHS packet sent in **inquiry response** state, the DCI (see Section 5.4) is used. For all other packets, the UAP of the master is used.

   The 8 bits are loaded into the 8 least significant (left-most) positions of the LFSR circuit, see Figure 5.10 on page 76. The other 8 bits are at the same time reset to zero. Subsequently, the CRC code is calculated over the information. Then the CRC code is appended to the information; the UAP (or DCI) is disregarded. At the receive side, the CRC circuitry is in the same way initialized with the 8-bit UAP (DCI) before the received information is checked. More information can be found in Section 5.4 on page 73.

4.6 PACKET SUMMARY

A summary of the packets and their characteristics is shown in Table 4.9, Table 4.10 and Table 4.11. The user payload represents the packet payload excluding FEC, CRC, and payload header.

Type	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate	Asymmetric Max. Rate
ID	na	na	na	na	na
NULL	na	na	na	na	na
POLL	na	na	na	na	na
FHS	18	2/3	yes	na	na

Table 4.9: Link control packets

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (kb/s)	Asymmetric Max. Rate (kb/s)	
						Forward	Reverse
DM1	1	0-17	2/3	yes	108.8	108.8	108.8
DH1	1	0-27	no	yes	172.8	172.8	172.8
DM3	2	0-121	2/3	yes	258.1	387.2	54.4
DH3	2	0-183	no	yes	390.4	585.6	86.4
DM5	2	0-224	2/3	yes	286.7	477.8	36.3
DH5	2	0-339	no	yes	433.9	723.2	57.6
AUX1	1	0-29	no	no	185.6	185.6	185.6

Table 4.10: ACL packets

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (kb/s)
HV1	na	10	1/3	no	64.0
HV2	na	20	2/3	no	64.0
HV3	na	30	no	no	64.0
DV*	1 D	10+(0-9) D	2/3 D	yes D	64.0+57.6 D

Table 4.11: SCO packets

*. Items followed by 'D' relate to data field only.

5 ERROR CORRECTION

There are three error correction schemes defined for Bluetooth:

- 1/3 rate FEC
- 2/3 rate FEC
- ARQ scheme for the data

The purpose of the FEC scheme on the data payload is to reduce the number of retransmissions. However, in a reasonable error-free environment, FEC gives unnecessary overhead that reduces the throughput. Therefore, the packet definitions given in Section 4 have been kept flexible to use FEC in the payload or not, resulting in the **DM** and **DH** packets for the ACL link and the **HV** packets for the SCO link. The packet header is always protected by a 1/3 rate FEC; it contains valuable link information and should be able to sustain more bit errors.

Correction measures to mask errors in the voice decoder are not included in this section. This matter is discussed in section Section 12.3 on page 142.

5.1 FEC CODE: RATE 1/3

A simple 3-times repetition FEC code is used for the header. The repetition code is implemented by repeating the bit three times, see the illustration in Figure 5.1 on page 67. The 3-bit repetition code is used for the entire header, and also for the voice field in the **HV1** packet.

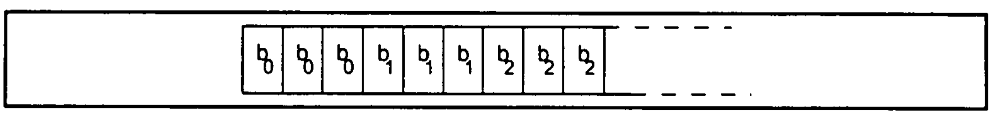


Figure 5.1: Bit-repetition encoding scheme.

5.2 FEC CODE: RATE 2/3

The other FEC scheme is a (15,10) shortened Hamming code. The generator polynomial is $g(D) = (D+1)(D^4+D+1)$. This corresponds to 65 in octal notation. The LFSR generating this code is depicted in Figure 5.2 on page 68. Initially all register elements are set to zero. The 10 information bits are sequentially fed into the LFSR with the switches S1 and S2 set in position 1. Then, after the final input bit, the switches S1 and S2 are set in position 2, and the five parity bits are shifted out. The parity bits are appended to the information bits. Consequently, each block of 10 information bits is encoded into a 15 bit codeword. This code can correct all single errors and detect all double errors in each codeword. This 2/3 rate FEC is used in the **DM** packets, in the data field of the **DV** packet, in the **FHS** packet, and in the **HV2** packet. Since the encoder operates with information segments of length 10, tail bits with

value zero may have to be appended after the CRC bits. The total number of bits to encode, i.e., payload header, user data, CRC, and tail bits, must be a multiple of 10. Thus, the number of tail bits to append is the least possible that achieves this (i.e., in the interval 0...9). These tail bits are not included in the payload length indicator.

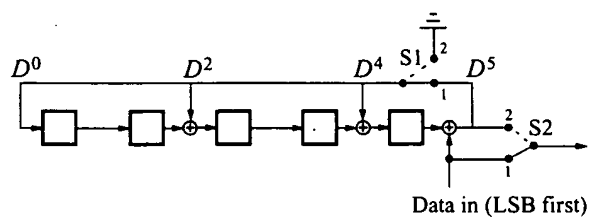


Figure 5.2: LFSR generating the (15,10) shortened Hamming code.

5.3 ARQ SCHEME

With an automatic repeat request scheme, **DM, DH** and the data field of **DV** packets are transmitted and retransmitted until acknowledgement of a successful reception is returned by the destination (or timeout is exceeded). The acknowledgement information is included in the header of the return packet, so-called piggy-backing. To determine whether the payload is correct or not, a cyclic redundancy check (CRC) code is added to the packet. The ARQ scheme only works on the payload in the packet (only that payload which has a CRC). The packet header and the voice payload are not protected by the ARQ scheme.

5.3.1 Unnumbered ARQ

Bluetooth uses a fast, unnumbered acknowledgment scheme: an ACK (ARQN=1) or a NAK (ARQN=0) is returned in response to the receipt of previously received packet. The slave will respond in the slave-to-master slot directly following the master-to-slave slot; the master will respond at the next event it will address the same slave (the master may have addressed other slaves between the last received packet from the considered slave and the master response to this packet). For a packet reception to be successful, at least the HEC must check. In addition, the CRC must check if present.

At the start of a new connection which may be the result of a page, page scan, master-slave switch or unpark, the master sends a POLL packet to verify the connection. In this packet the master initializes the ARQN bit to NAK. The response packet sent by the slave also has the ARQN bit set to NAK. The subsequent packets use the following rules.

The ARQ bit is affected by data packets containing CRC and empty slots only. As shown in Fig. 5.3 on page 70, upon successful reception of a CRC packet, the ARQN bit is set to ACK. If, in any receive slot in the slave or in a receive

slot following transmission of a packet in the master, no access code is detected, and the HEC check or the CRC check of a CRC packet fails, then the ARQN bit is set to NAK. Packets that have correct HEC but that are addressed to other slaves, or packets other than DH, DM, or DV packets, do not affect the ARQN bit. In these cases the ARQN bit is left as it was prior to reception of the packet. If a CRC packet with a correct header has the same SEQN as the previously received CRC packet, the ARQN bit is set to ACK and the payload is disregarded without checking the CRC.

The ARQ bit in the FHS packet is not meaningful. Contents of the ARQN bit in the FHS packet should not be checked.

Broadcast packets are checked on errors using the CRC, but no ARQ scheme is applied. Broadcast packets are never acknowledged.

Inactive connection modes HOLD and SNIFF do not affect the ARQN scheme. After return from these modes, packets will continue using values from before the start of hold/sniff modes.

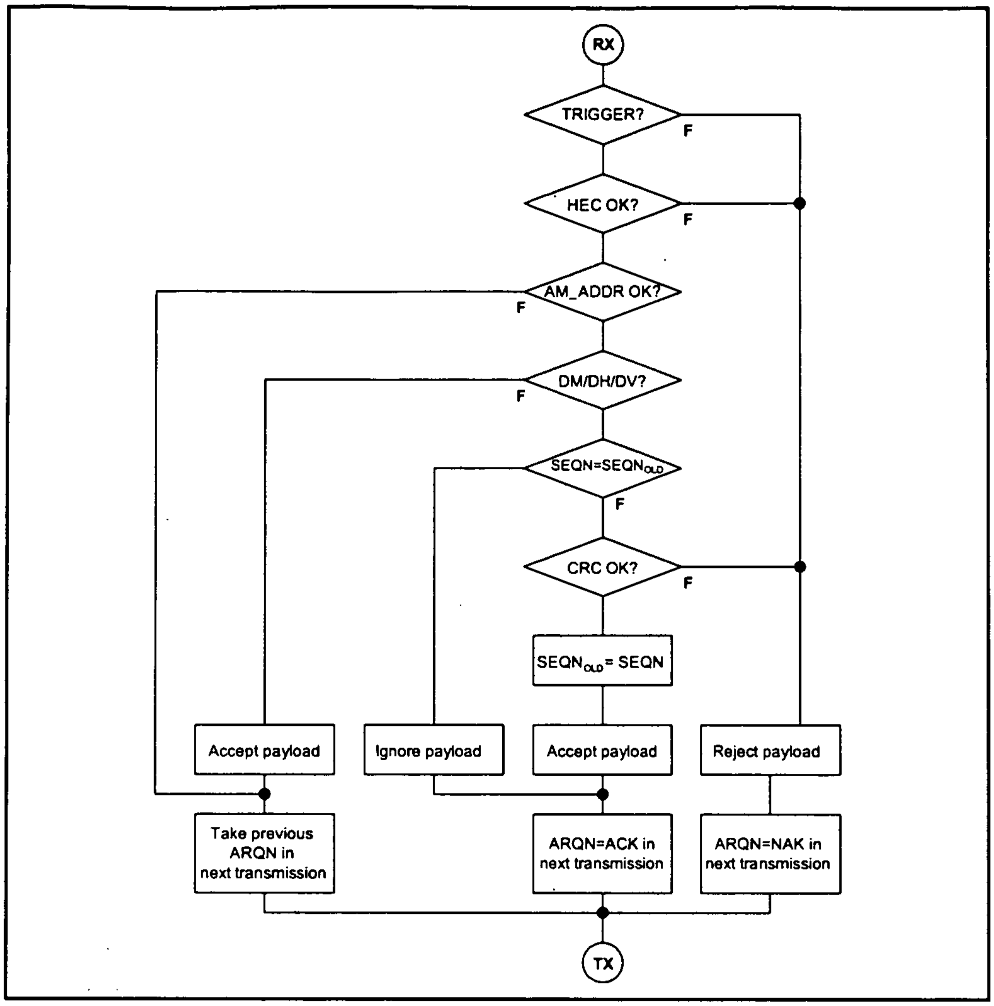


Figure 5.3: Receive protocol for determining the ARQN bit.

5.3.2 Retransmit filtering

The data payload is retransmitted until a positive acknowledgment is received (or a timeout is exceeded). A retransmission is carried out either because the packet transmission itself failed, or because the piggy-backed acknowledgment transmitted in the return packet failed (note that the latter has a lower failure probability since the header is more heavily coded). In the latter case, the destination keeps receiving the same payload over and over again. In order to filter out the retransmissions in the destination, the SEQN bit is added in the header. Normally, this bit is alternated for every new CRC data payload transmission. In case of a retransmission, this bit is not changed. So the destination can compare the SEQN bit with the previous SEQN value. If different, a new data payload has arrived; otherwise it is the same data payload and can be discarded. Only new data payloads are transferred to the link manager. Note that CRC data payloads can be carried only by **DM, DH** or **DV** packets.

At the start of a new connection which may be the result of a page, page scan, master slave switch or unpark, the master sends a POLL packet to verify the connection. The slave responds with a packet. The SEQN bit of the first CRC data packet, on both the master and the slave sides, is set to 1. The subsequent packets use the rules given below.

The SEQN bit is affected only by the CRC data packets as shown in Figure 5.4. It is inverted every time a new CRC data packet is sent. The CRC data packet is retransmitted with the same SEQN number until an ACK is received or the packet is flushed. When an ACK is received, the SEQN bit is inverted and a new payload is sent. When the packet is flushed (see Section 5.3.3 on page 71), a new payload is sent. The SEQN bit is not necessarily inverted. However, if an ACK is received before the new packet is sent, the SEQN bit is inverted. This procedure prevents loss of the first packet of a message (after the **flush** command has been given) due to the retransmit filtering.

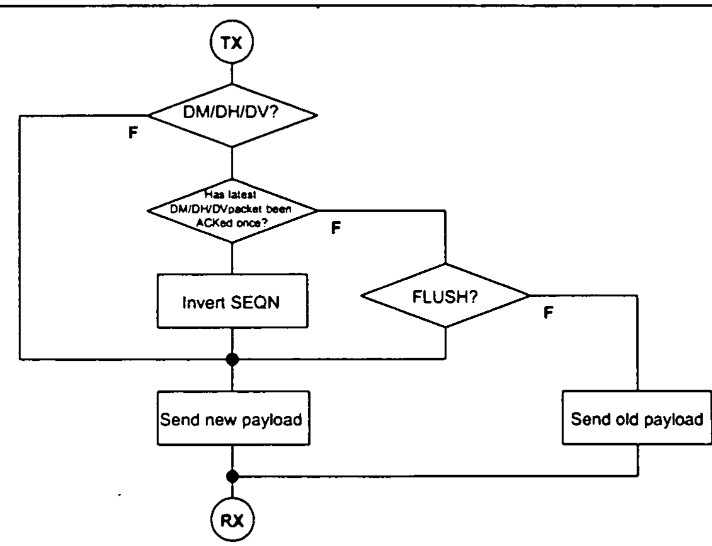


Figure 5.4: Retransmit filtering for packets with CRC.

The SEQN bit in the FHS packet is not meaningful. This bit can be set to any value. Contents of the SEQN bit in the FHS packet should not be checked. During transmission of all other packets the SEQN bit remains the same as it was in the previous packet.

Inactive connection modes HOLD and SNIFF do not affect the SEQN scheme. After return from these modes, packets will continue using values from before the start of hold/sniff modes.

5.3.3 Flushing payloads

The ARQ scheme can cause variable delay in the traffic flow since retransmissions are inserted to assure error-free data transfer. For certain communication

links, only a limited amount of delay is allowed: retransmissions are allowed up to a certain limit at which the current payload must be disregarded and the next payload must be considered. This data transfer is indicated as **isochronous traffic**. This means that the retransmit process must be overruled in order to continue with the next data payload. Aborting the retransmit scheme is accomplished by *flushing* the old data and forcing the Bluetooth controller to take the next data instead.

Flushing results in loss of remaining portions of an L2CAP message. Therefore, the packet following the flush will have a start packet indication of L_CH = 10 in the payload header for the next L2CAP message. This informs the destination of the flush. (see Section 4.5). Flushing will not necessarily result in a change in the SEQN bit value, see the previous section.

5.3.4 Multi-slave considerations

In case of a piconet with multiple slaves, the master carries out the ARQ protocol independently to each slave.

5.3.5 Broadcast packets

Broadcast packets are packets transmitted by the master to all the slaves simultaneously. A broadcast packet is indicated by the all-zero AM_ADDR (note; the FHS packet is the only packet which may have an all-zero address but is not a broadcast packet). Broadcast packets are not acknowledged (at least not at the LC level).

Since broadcast messages are not acknowledged, each broadcast packet is repeated for a fixed number of times. A broadcast packet is repeated N_{BC} times before the next broadcast packet of the same broadcast message is repeated, see Figure 5.5 on page 73.

Broadcast packets with a CRC have their own sequence number. The SEQN of the first broadcast packet with a CRC is set to SEQN = 1 by the master and it is inverted for each new broadcast packet with CRC thereafter. Broadcast packets without a CRC have no influence on the sequence number. The slave accepts the SEQN of the first broadcast packet it receives in a connection and checks for change in SEQN for consequent broadcast packets. Since there is no acknowledgement of broadcast messages and there is no end packet indication, it is important to receive the start packets correctly. To ensure this, repetitions of the broadcast packets that are L2CAP start packets and LMP packets will not be filtered out. These packets are indicated by L_CH=1X in the payload header as explained in section 4.5 on page 62. Only repetitions of the L2CAP continuation packets will be filtered out.

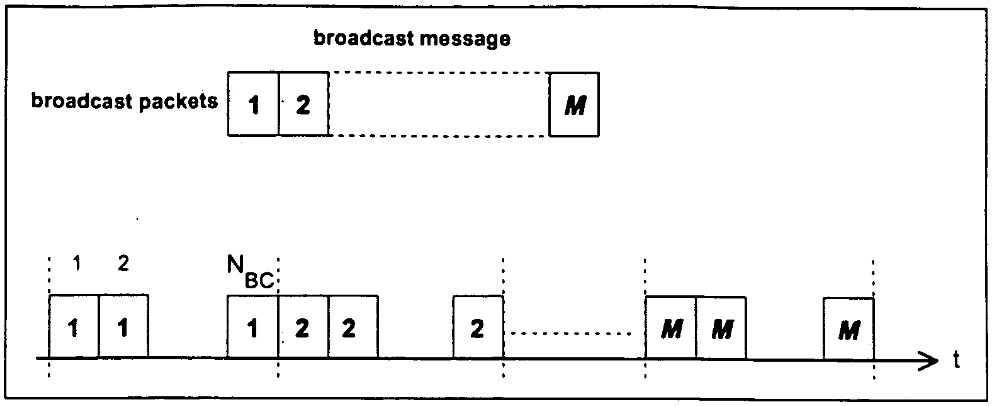


Figure 5.5: Broadcast repetition scheme

5.4 ERROR CHECKING

We can check the packet for errors or wrong delivery using the channel access code, the HEC in the header, and the CRC in the payload. At packet reception, first the access code is checked. Since the 64-bit sync word in the channel access code is derived from the 24-bit master LAP, this checks if the LAP is correct, and prevents the receiver from accepting a packet of another piconet.

The HEC and CRC are used to check both on errors and on a wrong address: to increase the address space with 8 bits, the UAP is normally included in the HEC and CRC checks. Then, even when a packet with the same access code – i.e., an access code of a device owning the same LAP but different UAP – passes the access code test, it will be discarded after the HEC and CRC tests when the UAP bits do not match. However, there is an exception where no common UAP is available in the transmitter and receiver. This is the case when the HEC and CRC are generated for the FHS packet in **inquiry response** state. In this case the default check initialization (DCI) value is used. The DCI is defined to be `0x00` (hexadecimal).

The generation and check of the HEC and CRC are summarized in Figure 5.8 on page 75 and Figure 5.11 on page 76. Before calculating the HEC or CRC, the shift registers in the HEC/CRC generators are initialized with the 8-bit UAP (or DCI) value. Then the header and payload information is shifted into the HEC and CRC generators, respectively (with the LSB first).

The HEC generating LFSR is depicted in Figure 5.6 on page 74. The generator polynomial is $g(D) = (D+1)(D^7+D^4+D^3+D^2+1) = D^8+D^7+D^5+D^2+D+1$. Initially this circuit is pre-loaded with the 8-bit UAP such that the LSB of the UAP (denoted UAP_0) goes to the left-most shift register element, and, UAP_7 goes to the right-most element. The initial state of the HEC LFSR is depicted in Figure 5.7 on page 75. Then the data is shifted in with the switch S set in position 1. When the last data bit has been clocked into the LFSR, the switch S is set in position 2, and, the HEC can be read out from the register. The LFSR bits

are read out from right to left (i.e., the bit in position 7 is the first to b transmitted, followed by the bit in position 6, etc.).

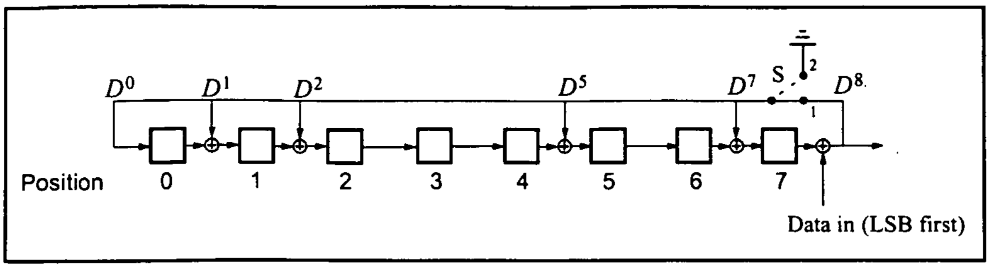


Figure 5.6: The LFSR circuit generating the HEC.

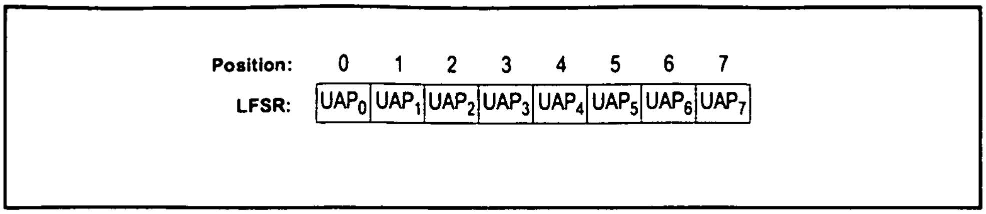


Figure 5.7: Initial state of the HEC generating circuit.

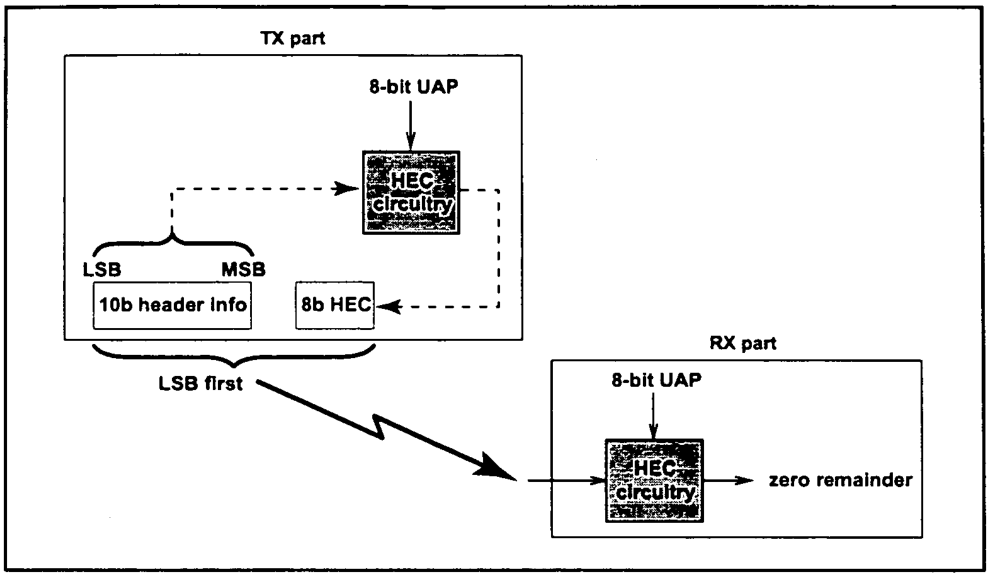


Figure 5.8: HEC generation and checking.

The 16 bit LFSR for the CRC is constructed similarly using the CRC-CCITT generator polynomial $g(D) = D^{16} + D^{12} + D^5 + 1$ (see Figure 5.9 on page 75). For this case, the 8 left-most bits are initially loaded with the 8-bit UAP (UAP_0 to the left and UAP_7 to the right) while the 8 right-most bits are reset to zero. The initial state of the 16 bit LFSR is depicted in Figure 5.10 on page 76. The switch S is set in position 1 while the data is shifted in. After the last bit has entered the LFSR, the switch is set in position 2, and, the register's contents are transmitted, from right to left (i.e., starting with position 15, then position 14, etc.).

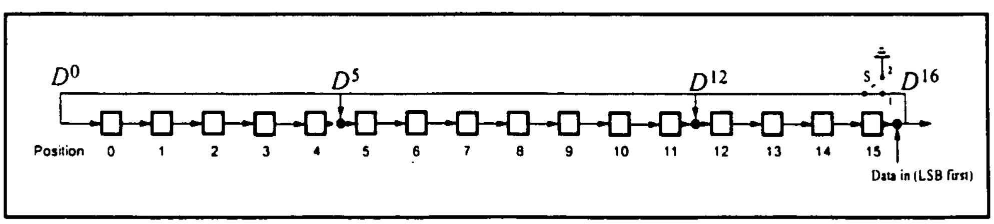


Figure 5.9: The LFSR circuit generating the CRC.

Position:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LFSR:	UAP_0	UAP_1	UAP_2	UAP_3	UAP_4	UAP_5	UAP_6	UAP_7	0	0	0	0	0	0	0	0

Figure 5.10: Initial state of the CRC generating circuit.

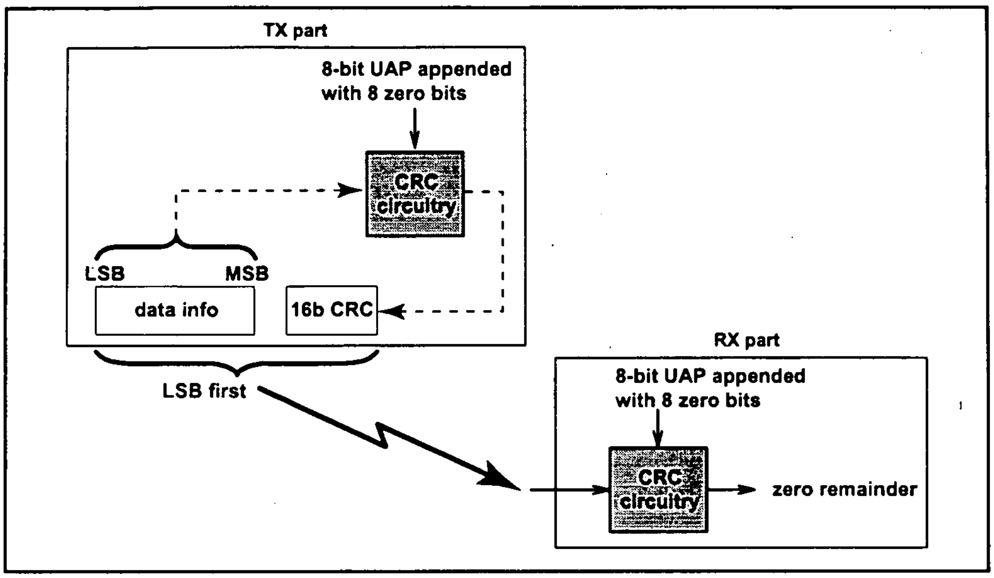


Figure 5.11: CRC generation and checking

6 LOGICAL CHANNELS

In the Bluetooth system, five logical channels are defined:

- LC control channel
- LM control channel
- UA user channel
- UI user channel
- US user channel

The control channels LC and LM are used at the link control level and link manager level, respectively. The user channels UA, UI, and US are used to carry asynchronous, isochronous, and synchronous user information, respectively. The LC channel is carried in the packet header; all other channels are carried in the packet payload. The LM, UA, and UI channels are indicated in the L_CH field in the payload header. The US channel is carried by the SCO link only; the UA and UI channels are normally carried by the ACL link; however, they can also be carried by the data in the DV packet on the SCO link. The LM channel can be carried either by the SCO or the ACL link.

6.1 LC CHANNEL (Link Control)

The LC control channel is mapped onto the packet header. This channel carries low level link control information like ARQ, flow control, and payload characterization. The LC channel is carried in every packet except in the **ID** packet which has no packet header.

6.2 LM CHANNEL (Link Manager)

The LM control channel carries control information exchanged between the link managers of the master and the slave(s). Typically, the LM channel uses protected **DM** packets. The LM channel is indicated by the L_CH code 11 in the payload header.

6.3 UA/UI CHANNEL (User Asynchronous/Isochronous Data)

The UA channel carries L2CAP transparent asynchronous user data. This data may be transmitted in one or more baseband packets. For fragmented messages, the start packet uses an L_CH code of 10 in the payload header. Remaining continuation packets use L_CH code 01. If there is no fragmentation, all packets use the L2CAP start code 10.

Isochronous data channel is supported by timing start packets properly at higher levels. At the baseband level, the L_CH code usage is the same as the UA channel.

6.4 US CHANNEL (Us r Synchronous Data)

The US channel carries transparent synchronous user data. This channel is carried over the SCO link.

6.5 CHANNEL MAPPING

The LC channel is mapped onto the packet header. All other channels are mapped onto the payload. The US channel can only be mapped onto the SCO packets. All other channels are mapped on the ACL packets, or possibly the SCO **DV** packet. The LM, UA, and UI channels may interrupt the US channel if it concerns information of higher priority.

7 DATA WHITENING

Before transmission, both the header and the payload are scrambled with a data whitening word in order to randomize the data from highly redundant patterns and to minimize DC bias in the packet. The scrambling is performed prior to the FEC encoding.

At the receiver, the received data is descrambled using the same whitening word generated in the recipient. The descrambling is performed after FEC decoding.

The whitening word is generated with the polynomial $g(D) = D^7 + D^4 + 1$ (i.e., `221` in octal representation) and is subsequently EXORed with the header and the payload. The whitening word is generated with the linear feedback shift register shown in Figure 7.1 on page 79. Before each transmission, the shift register is initialized with a portion of the master Bluetooth clock, CLK_{6-1}, extended with an MSB of value one. This initialization is carried out with CLK_1 written to position 0, CLK_2 written to position 1, etc. An exception forms the FHS packet sent during frequency hop acquisition, where initialization of the whitening register is carried out differently. Instead of the master clock, the X-input used in the **inquiry** or **page response** (depending on current state) routine is used, see Table 11.3 and Table 11.4 for the 79-hop and 23-hop systems, respectively. In case of a 79-hop system, the 5-bit values is extended with two MSBs of value one. In case of a 23-hop system, the 4-bit value is extended with three bits; the two MSBs are set to one and the third most significant bit is set to zero. During register initialization, the LSB of X (i.e., X_0) is written to position 0, X_1 is written to position 1, etc.

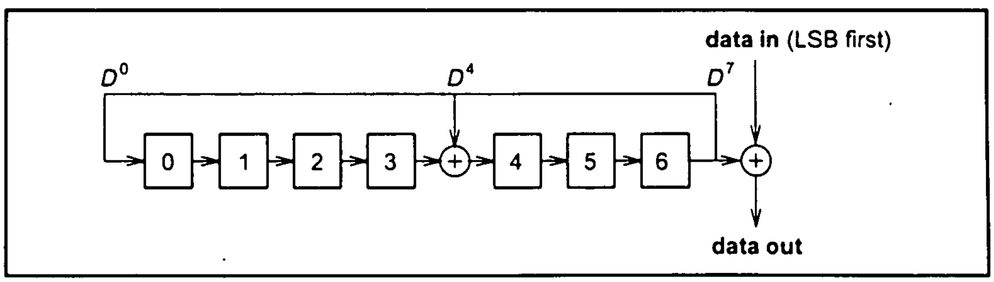


Figure 7.1: Data whitening LFSR.

After initialization, the packet header and the payload (including the CRC) are scrambled. The payload whitening continues from the state the whitening LFSR had at the end of HEC. There is no re-initialization of the shift register between packet header and payload. The first bit of the "Data In" sequence is the LSB of the packet header.

8 TRANSMIT/RECEIVE ROUTINES

This section describes the way to use the packets as defined in Section 4 in order to support the traffic on the ACL and SCO links. Both single-slave and multi-slave configurations are considered. In addition, the use of buffers for the TX and RX routines are described.

The TX and RX routines described in sections 8.1 and 8.2 are of an informative character only. The final implementation may be carried out differently.

8.1 TX ROUTINE

The TX routine is carried out separately for each ACL link and each SCO link. Figure 8.1 on page 81 shows the ACL and SCO buffers as used in the TX routine. In this figure, only a single TX ACL buffer and a single TX SCO buffer are shown. In the master, there is a separate TX ACL buffer for each slave. In addition there may be one or more TX SCO buffers for each SCO slave (different SCO links may either reuse the same TX SCO buffer, or each have their own TX SCO buffer). Each TX buffer consists of two FIFO registers: one **current** register which can be accessed and read by the Bluetooth controller in order to compose the packets, and one **next** register that can be accessed by the Bluetooth Link Manager to load new information. The positions of the switches S1 and S2 determine which register is current and which register is next; the switches are controlled by the Bluetooth Link Controller. The switches at the input and the output of the FIFO registers can never be connected to the same register simultaneously.

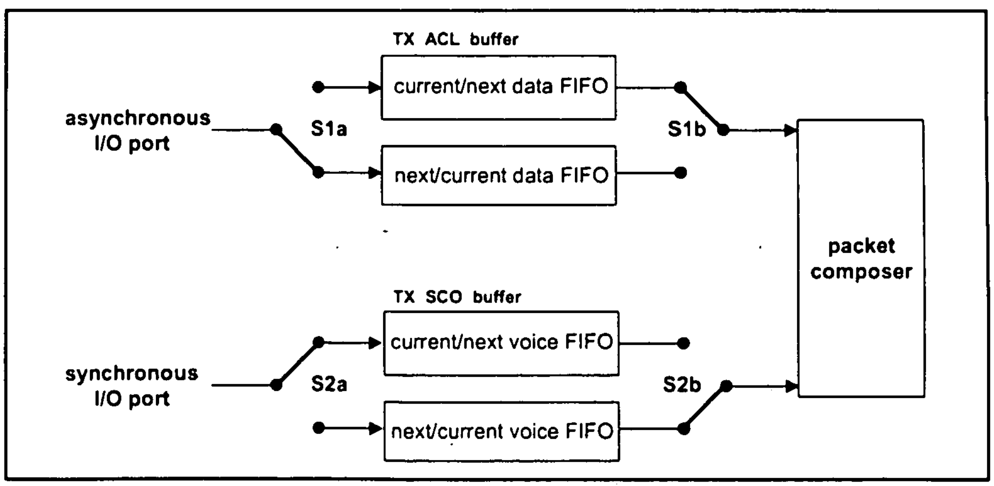


Figure 8.1: Functional diagram of TX buffering.

Of the packets common on the ACL and SCO links (**ID**, **NULL**, **POLL**, **FHS**, **DM1**) only the **DM1** packet carries a payload that is exchanged between the Link Controller and the Link Manager; this common packet makes use of the

ACL buffer. All ACL packets make use of the ACL buffer. All SCO packets make use of the SCO buffer except for the **DV** packet where the voice part is handled by the SCO buffer and the data part is handled by the ACL buffer. In the next sections, the operation for ACL traffic, SCO traffic, and combined data-voice traffic on the SCO link will be considered.

8.1.1 ACL traffic

In the case of pure (asynchronous) data, only the TX ACL buffer in Figure 8.1 on page 81 has to be considered. In this case, only packet types **DM** or **DH** are used, and these can have different lengths. The length is indicated in the payload header. The selection of high-rate data or medium-rate data shall depend on the quality of the link. When the quality is good, the FEC in the data payload can be omitted, resulting in a **DH** packet. Otherwise, **DM** packets must be used.

The default TYPE in pure data traffic is **NULL**. This means that, if there is no data to be sent (the data traffic is asynchronous, and therefore pauses occur in which no data is available) or no slaves need to be polled, **NULL** packets are sent instead – in order to send link control information to the other Bluetooth unit (e.g. ACK/STOP information for received data). When no link control information is available either (no need to acknowledge and/or no need to stop the RX flow) no packet is sent at all.

The TX routine works as follows. The Bluetooth Link Manager loads new data information in the register to which the switch S1a points. Next, it gives a **flush** command to the Bluetooth Link Controller, which forces the switch S1 to change (both S1a and S1b switch in synchrony). When the payload needs to be sent, the packet composer reads the current register and, depending on the packet TYPE, builds a payload which is appended to the channel access code and the header and is subsequently transmitted. In the response packet (which arrives in the following RX slot if it concerned a master transmission, or may be postponed until some later RX slot if it concerned a slave transmission), the result of the transmission is reported back. In case of an ACK, the switch S1 changes position; if a NAK (explicit or implicit) is received instead, the switch S1 will not change position. In that case, the same payload is retransmitted at the next TX occasion.

As long as the Link Manager keeps loading the registers with new information, the Bluetooth Link Controller will automatically transmit the payload; in addition, retransmissions are performed automatically in case of errors. The Link Controller will send **NULL** or nothing when no new data is loaded. If no new data has been loaded in the **next** register, during the last transmission, the packet composer will be pointing to an empty register after the last transmission has been acknowledged and the **next** register becomes the **current** register. If new data is loaded in the **next** register, a **flush** command is required to switch the S1 switch to the proper register. As long as the Link Manager keeps loading the data and type registers before each TX slot, the data is automatically processed by the Link Controller since the S1 switch is controlled by the

ACK information received in response. However, if the traffic from the Link Manager is interrupted once and a default packet is sent instead, a **flush** command is required to continue the flow in the Link Controller.

The **flush** command can also be used in case of time-bounded (isochronous) data. In case of a bad link, many retransmission are necessary. In certain applications, the data is time-bounded: if a payload is retransmitted all the time because of link errors, it may become outdated, and the system might decide to continue with more recent data instead and skip the payload that does not come through. This is accomplished by the **flush** command as well. With the **flush**, the switch S1 is forced to change and the Link Controller is forced to consider the next data payload and overrules the ACK control.

8.1.2 SCO traffic

In case of an SCO link, we only use **HV** packet types. The synchronous port continuously loads the **next** register in the SCO buffer. The S2 switches are changed according to the T_{sco} interval. This T_{sco} interval is negotiated between the master and the slave at the time the SCO link is established.

For each new SCO slot, the packet composer reads the **current** register after which the S2 switch is changed. If the SCO slot has to be used to send control information with high priority concerning a control packet between the master and the considered SCO slave, or a control packet between the master and any other slave, the packet composer will discard the SCO information and use the control information instead. This control information must be sent in a DM1 packet. Data or link control information can also be exchanged between the master and the SCO slave by using the **DV** or **DM1** packets. Any ACL type of packet can be used to sent data or link control information to any other ACL slave. This is discussed next.

8.1.3 Mixed data/voice traffic

In Section 4.4.2 on page 58, a **DV** packet has been defined that can support both data and voice simultaneously on a single SCO link. When the TYPE is **DV**, the Link Controller reads the data register to fill the data field and the voice register to fill the voice field. Thereafter, the switch S2 is changed. However, the position of S1 depends on the result of the transmission like on the ACL link: only if an ACK has been received will the S1 switch change its position. In each **DV** packet, the voice information is new, but the data information might be retransmitted if the previous transmission failed. If there is no data to be sent, the SCO link will automatically change from **DV** packet type to the current **HV** packet type used before the mixed data/voice transmission. Note that a **flush** command is required when the data stream has been interrupted and new data has arrived.

Combined data-voice transmission can also be accomplished by using separate ACL links in addition to the SCO link(s) if channel capacity permits this.

8.1.4 Default packet types

On the ACL links, the default type is always **NULL** both for the master and the slave. This means that if no user information needs to be send, either a **NULL** packet is sent if there is **ACK** or **STOP** information, or no packet is sent at all. The **NULL** packet can be used by the master to allocate the next slave-to-master slot to a certain slave (namely the one addressed). However, the slave is not forced to respond to the **NULL** packet from the master. If the master requires a response, it has to send a **POLL** packet.

The SCO packet type is negotiated at the LM level when the SCO link is established. The agreed packet type is also the default packet type for the SCO slots.

8.2 RX ROUTINE

The RX routine is carried out separately for the ACL link and the SCO link. However, in contrast to the master TX ACL buffer, a single RX buffer is shared among all slaves. For the SCO buffer, it depends how the different SCO links are distinguished whether extra SCO buffers are required or not. Figure 8.2 on page 84 shows the ACL and SCO buffers as used in the RX routine. The RX ACL buffer consists of two FIFO registers: one register that can be accessed and loaded by the Bluetooth Link Controller with the payload of the latest RX packet, and one register that can be accessed by the Bluetooth Link Manager to read the previous payload. The RX SCO buffer also consists of two FIFO registers: one register which is filled with newly arrived voice information, and one register which can be read by the voice processing unit.

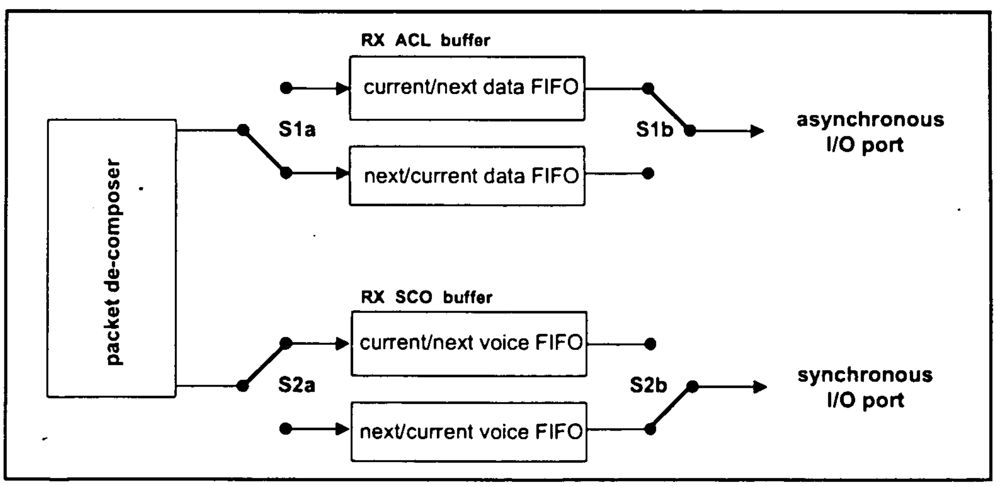


Figure 8.2: Functional diagram of RX buffering

Since the TYPE indication in the header of the received packet indicates whether the payload contains data and/or voice, the packet de-composer can automatically direct the traffic to the proper buffers. The switch S1 changes

every time the Link Manager has read the old register. If the next payload arrives before the RX register is emptied, a STOP indication must be included in the packet header of the next TX packet that is returned. The STOP indication is removed again as soon as the RX register is emptied. The SEQN field is checked before a new ACL payload is stored into the ACL register (flush indication in L_CH and broadcast messages influence the interpretation of the SEQN field see Section 5.3 on page 68).

The S2 switch is changed every T_{SCO}. If – due to errors in the header – no new voice payload arrives, the switch still changes. The voice processing unit then has to process the voice signal to account for the missing speech parts.

8.3 FLOW CONTROL

Since the RX ACL buffer can be full while a new payload arrives, flow control is required. As was mentioned earlier, the header field FLOW in the return TX packet can use STOP or GO in order to control the transmission of new data.

8.3.1 Destination control

As long as data cannot be received, a STOP indication is transmitted which is automatically inserted by the Link Controller into the header of the return packet. STOP is returned as long as the RX ACL buffer is not emptied by the Link Manager. When new data can be accepted again, the GO indication is returned. GO is the default value. Note that all packet types not including data can still be received. Voice communication for example is not affected by the flow control. Also note that although a Bluetooth unit cannot receive new information, it can still continue to transmit information: the flow control is separate for each direction.

8.3.2 Source control

On the reception of a STOP signal, the Link Controller will automatically switch to the default packet type. The current TX ACL buffer status is frozen. Default packets are sent as long as the STOP indication is received. When no packet is received, GO is assumed implicitly. Note that the default packets contain link control information (in the header) for the receive direction (which may still be open) and may contain voice (**HV** packets). When a GO indication is received, the Link Controller resumes to transmit the data as is present in the TX ACL buffers.

In a multi-slave configuration, only the transmission to the slave that issued the STOP signal is stalled. This means that the previously described routine implemented in the master only concerns the TX ACL buffer that corresponds to the slave that cannot accept data momentarily.

8.4 BITSTREAM PROCESSES

Before the user information is sent over the air interface, several bit manipulations are performed in the transmitter to increase reliability and security. To the packet header, an HEC is added, the header bits are scrambled with a whitening word, and FEC coding is applied. In the receiver, the inverse processes are carried out. Figure 8.3 on page 86 shows the processes carried out for the packet header both at the transmit and the receive side. All header bit processes are mandatory.

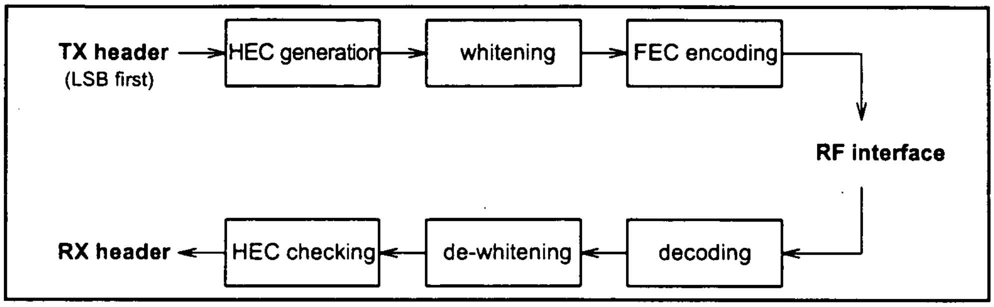


Figure 8.3: Header bit processes.

For the payload, similar processes are performed. It depends on the packet type, which processes are carried out. Figure 8.4 on page 86 shows the processes that may be carried out on the payload. In addition to the processes defined for the packet header, encryption can be applied on the payload. Only whitening and de-whitening, as explained in Section 7 on page 79, are mandatory for every payload; all other processes are optional and depend on the packet type and the mode enabled. In Figure 8.4 on page 86, optional processes are indicated by dashed blocks.

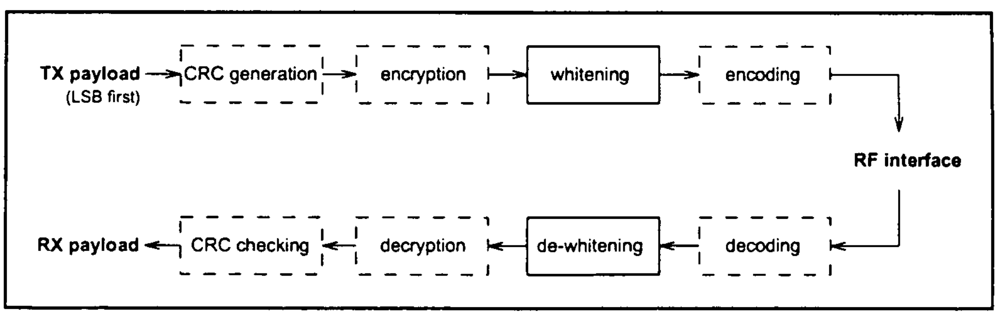


Figure 8.4: Payload bit processes.

9 TRANSMIT/RECEIVE TIMING

The Bluetooth transceiver applies a time-division duplex (TDD) scheme. This means that it alternately transmits and receives in a synchronous manner. It depends on the mode of the Bluetooth unit what the exact timing of the TDD scheme is. In the normal connection mode, *the master transmission shall always start at even numbered time slots (master CLK1=0) and the slave transmission shall always start at odd numbered time slots (master CLK1=1).* Due to packet types that cover more than a single slot, master transmission may continue in odd numbered slots and slave transmission may continue in even numbered slots.

All timing diagrams shown in this chapter are based on the signals as present at the antenna. The term "exact" when used to describe timing refers to an ideal transmission or reception and neglects timing jitter and clock frequency imperfections.

The average timing of master packet transmission must not drift faster than 20 ppm relative to the ideal slot timing of 625 µs. The instantaneous timing must not deviate more than 1 µs from the average timing. Thus, the absolute packet transmission timing t_k of slot boundary k must fulfill the equation:

$$t_k = \left(\sum_{i=1}^{k} (1 + d_i) T_N \right) + j_k + \text{offset}, \qquad \text{(EQ 1)}$$

where T_N is the nominal slot length (625 µs), j_k denotes jitter ($|j_k| \le 1$ µs) at slot boundary k, and, d_k, denotes the drift ($|d_k| \le 20$ ppm) within slot k. The jitter and drift may vary arbitrarily within the given limits for every slot, while "offset" is an arbitrary but fixed constant. For hold, park and sniff mode the drift and jitter parameters as described in Link Manager Protocol Section 3.9 on page 203 apply.

9.1 MASTER/SLAVE TIMING SYNCHRONIZATION

The piconet is synchronized by the system clock of the master. The master never adjusts its system clock during the existence of the piconet: it keeps an exact interval of Mx625 µs (where M is an even, positive integer larger than 0) between consecutive transmissions. The slaves adapt their native clocks with a timing offset in order to match the master clock. This offset is updated each time a packet is received from the master: by comparing the exact RX timing of the received packet with the estimated RX timing, the slaves correct the offset for any timing misalignments. Note that the slave RX timing can be corrected with any packet sent in the master-to-slave slot, since only the channel access code is required to synchronize the slave.

The slave TX timing shall be based on the most recent slave RX timing. The RX timing is based on the latest successful trigger during a master-to-slave slot. For ACL links, this trigger must have occurred in the master-to-slave slot directly pre-

ceding the current slave transmission; for SCO links, the trigger may have occurred several master-to-slave slots before since a slave is allowed to send an SCO packet even if no packet was received in the preceding master-to-slave slot. The slave shall be able to receive the packets and adjust the RX timing as long as the timing mismatch remains within the ± 10 μs uncertainty window.

The master TX timing is strictly related to the master clock.The master shall keep an exact interval of Mx1250 μs (where M is a positive integer larger than 0) between the start of successive transmissions; the RX timing is based on this TX timing with a shift of exactly Nx625 μs (where N is an odd, positive integer larger than 0). During the master RX cycle, the master will also use the $\pm 10\mu$ uncertainty window to allow for slave misalignments. The master will adjust the RX processing of the considered packet accordingly, but will **not** adjust its RX/TX timing for the following TX and RX cycles.

Timing behaviour may differ slightly depending on the current state of the unit. The different states are described in the next sections.

9.2 CONNECTION STATE

In the connection mode, the Bluetooth transceiver transmits and receives alternately, see Figure 9.1 on page 88 and Figure 9.2 on page 89. In the figures, only single-slot packets are shown as an example. Depending on the type and the payload length, the packet size can be up to 366 μs. Each RX and TX transmission is at a different hop frequency. For multi-slot packets, several slots are covered by the same packet, and the hop frequency used in the first slot will be used throughout the transmission.

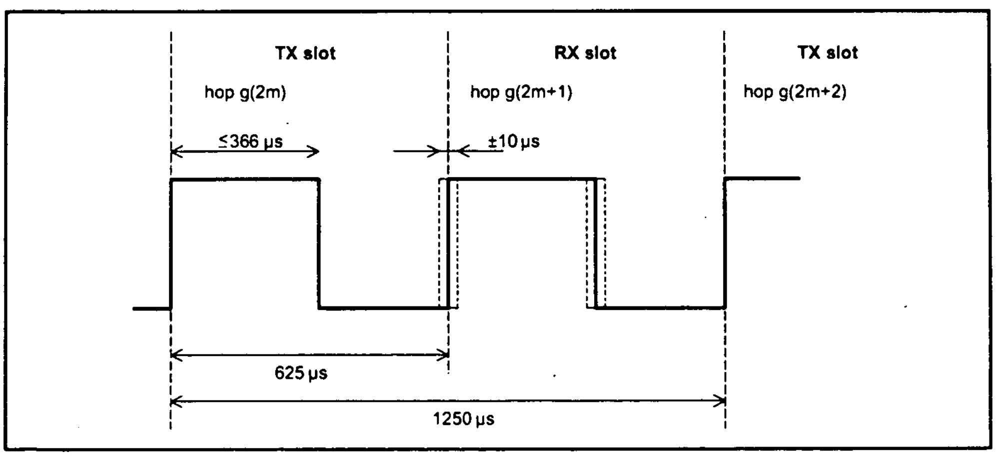


Figure 9.1: RX/TX cycle of Bluetooth master transceiver in normal mode for single-slot packets.

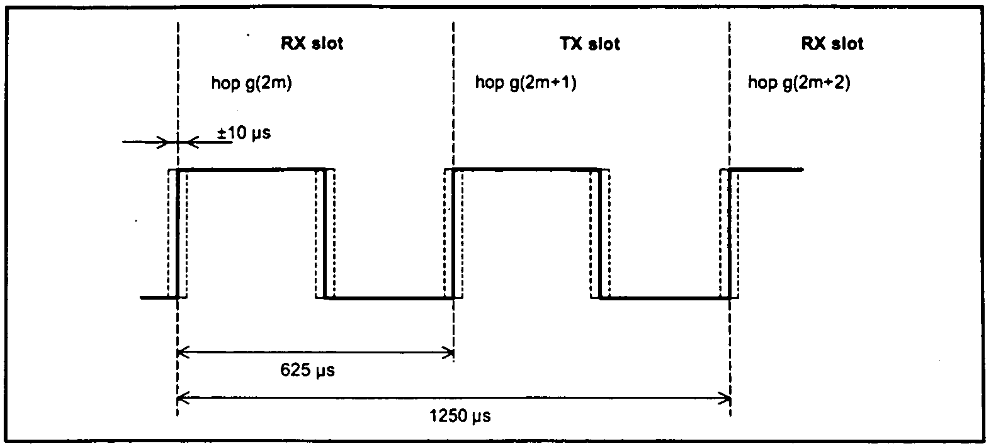


Figure 9.2: RX/TX cycle of Bluetooth slave transceiver in normal mode for single-slot packets.

The master TX/RX timing is shown in Figure 9.1 on page 88. In figures 9.1 through 9.6, f(k) is used for the frequencies of the page hopping sequence and f'(k) denotes the corresponding page response sequence frequencies. The channel hopping frequencies are indicated by g(m). After transmission, a return packet is expected Nx625 μs after the start of the TX burst where N is an odd, positive integer. N depends on the type of the transmitted packet. To allow for some time slipping, an uncertainty window is defined around the exact receive timing. During normal operation, the window length is 20 μs, which allows the RX burst to arrive up to 10 μs too early or 10 μs too late. During the beginning of the RX cycle, the access correlator searches for the correct channel access code over the uncertainty window. If no trigger event occurs, the receiver goes to sleep until the next RX event. If in the course of the search, it becomes apparent that the correlation output will never exceed the final threshold, the receiver may go to sleep earlier. If a trigger event does occur, the receiver remains open to receive the rest of the packet.

The current master transmission is based on the previous master transmission: it is scheduled Mx1250μs after the start of the previous master TX burst where M depends on the transmitted and received packet type. Note that the master TX timing is not affected by time drifts in the slave(s). If no transmission takes place during a number of consecutive slots, the master will take the TX timing of the latest TX burst as reference.

The slave's transmission is scheduled Nx625μs after the start of the slave's RX burst. If the slave's RX timing drifts, so will its TX timing. If no reception takes place during a number of consecutive slots, the slave will take the RX timing of the latest RX burst as reference.

9.3 RETURN FROM HOLD MODE

In the connection state, the Bluetooth unit can be placed in a **hold** mode, see Section 10.8 on page 112. In the **hold** mode, a Bluetooth transceiver neither transmits nor receives information. When returning to the normal operation after a **hold** mode in a slave Bluetooth unit, the slave must listen for the master before it may send information. In that case, the search window in the slave unit may be increased from ±10 μs to a larger value X μs as illustrated in Figure 9.3 on page 90. Note that only RX hop frequencies are used: the hop frequency used in the master-to-slave (RX) slot is also used in the uncertainty window extended into the preceding time interval normally used for the slave-to-master (TX) slot.

If the search window exceeds 625 μs, consecutive windows shall not be centered at the start of RX hops g(2m), g(2m+2), ... g(2m+2i) (where 'i' is an integer), but at g(2m), g(2m+4), ... g(2m+4i), or even at g(2m), g(2m+6), ...g(2m+6i) etc. to avoid overlapping search windows. The RX hop frequencies used shall correspond to the RX slot numbers.

It is recommended that single slot packets are used upon return from hold to minimize the synchronization time, especially after long hold periods that require search windows exceeding 625 μs.

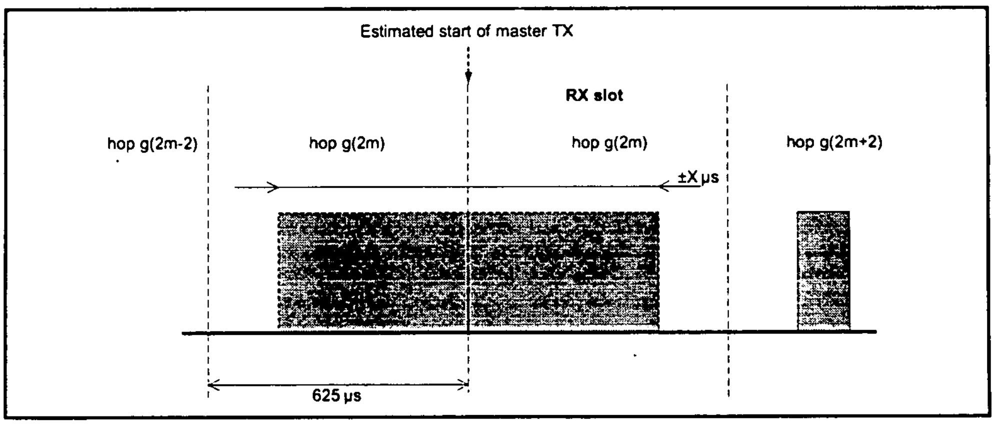


Figure 9.3: RX timing of slave returning from hold state.

9.4 PARK MODE WAKE-UP

The **park** mode is similar to the **hold** mode. A parked slave periodically wakes up to listen to beacons from the master and to re-synchronize its clock offset. As in the return from hold mode, a parked slave when waking up may increase the search window from ±10 μs to a larger value X μs as illustrated in Figure 9.3 on page 90.

9.5 PAGE STATE

In the page state, the master transmits the device access code (ID packet) corresponding to the slave to be connected, rapidly on a large number of different hop frequencies. Since the ID packet is a very short packet, the hop rate can be increased from 1600 hops/s to 3200 hops/s. In a single TX slot interval, the paging master transmits on two different hop frequencies. In a single RX slot interval, the paging transceiver listens on two different hop frequencies; see Figure 9.4 on page 91. During the TX slot, the paging unit sends an ID packet at the TX hop frequencies *f(k)* and *f(k+1)*. In the RX slot, it listens for a response on the corresponding RX hop frequencies *f'(k)* and *f'(k+1)*. The listening periods are exactly timed 625 μs after the corresponding paging packets, and include a ±10 μs uncertainty window.

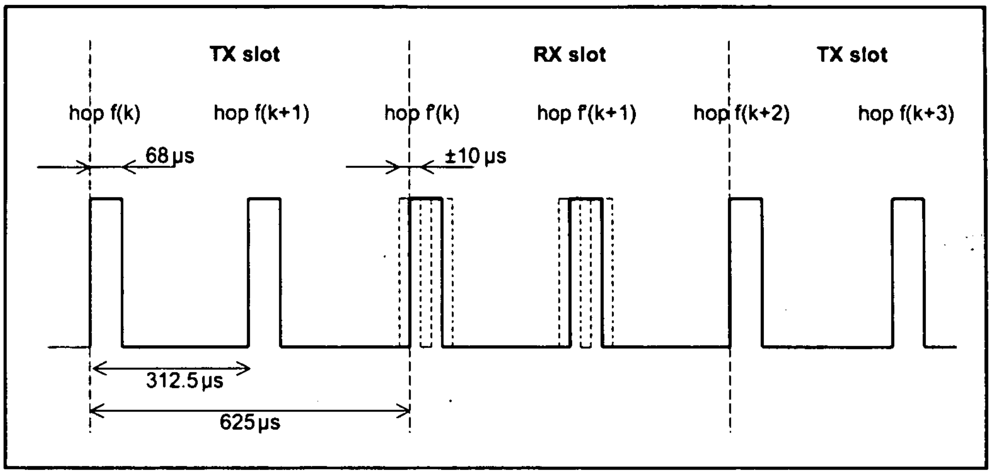


Figure 9.4: RX/TX cycle of Bluetooth transceiver in PAGE mode.

9.6 FHS PACKET

At connection setup and during a master-slave switch, an **FHS** packet is transferred from the master to the slave. This packet will establish the timing and frequency synchronization (see also Section 4.4.1.4 on page 56). After the slave unit has received the page message, it will return a response message which again consists of the ID packet and follows exactly 625 μs after the receipt of the page message. The master will send the FHS packet in the TX slot following the RX slot in which it received the slave response, according the RX/TX timing of the master. The time difference between the response and **FHS** message will depend on the timing of the page message the slave received. In Figure 9.5 on page 92, the slave receives the paging message sent **first** in the master-to-slave slot. It will then respond with an ID packet in the first half of the slave-to-master slot. The timing of the **FHS** packet is based on the timing of the page message sent first in the preceding master-to-slave slot: there is an exact 1250 μs delay between the first page message and the **FHS** packet. The packet is sent at the hop frequency *f(k+1)* which is the hop frequency following the hop frequency *f(k)* the page message was received in. In Figure 9.6 on page 92, the slave receives the paging message sent **s c-ondly** in the master-to-slave slot. It will then respond with an ID packet in the

second half of the slave-to-master slot exactly 625 µs after the receipt of the page message. The timing of the **FHS** packet is still based on the timing of the page message sent **first** in the preceding master-to-slave slot: there is an exact 1250 µs delay between the **first** page message and the **FHS** packet. The packet is sent at the hop frequency *f(k+2)* which is the hop frequency following the hop frequency *f(k+1)* the page message was received in.

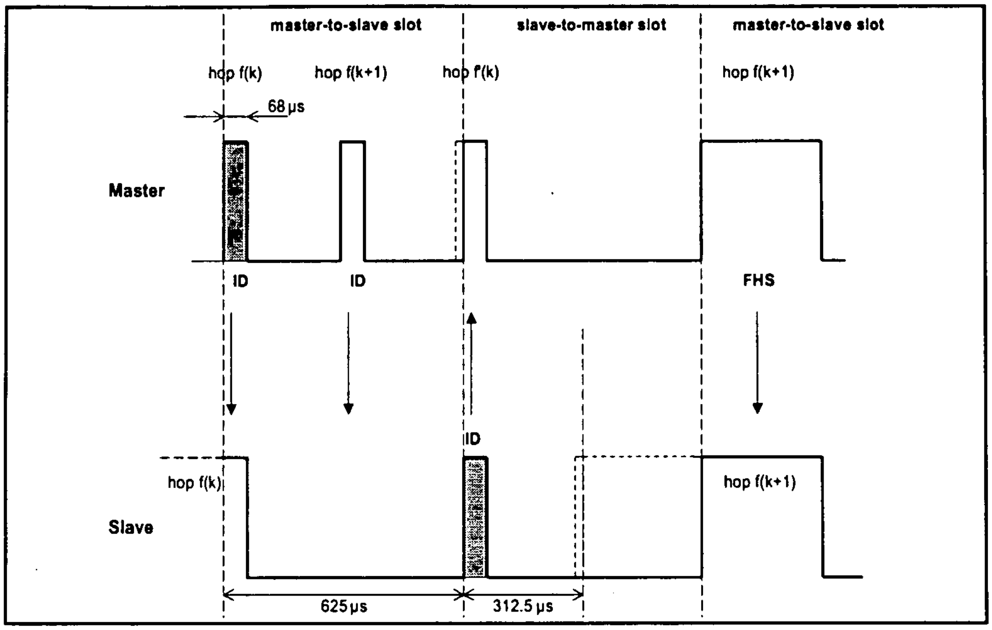


Figure 9.5: Timing of FHS packet on successful page in first half slot.

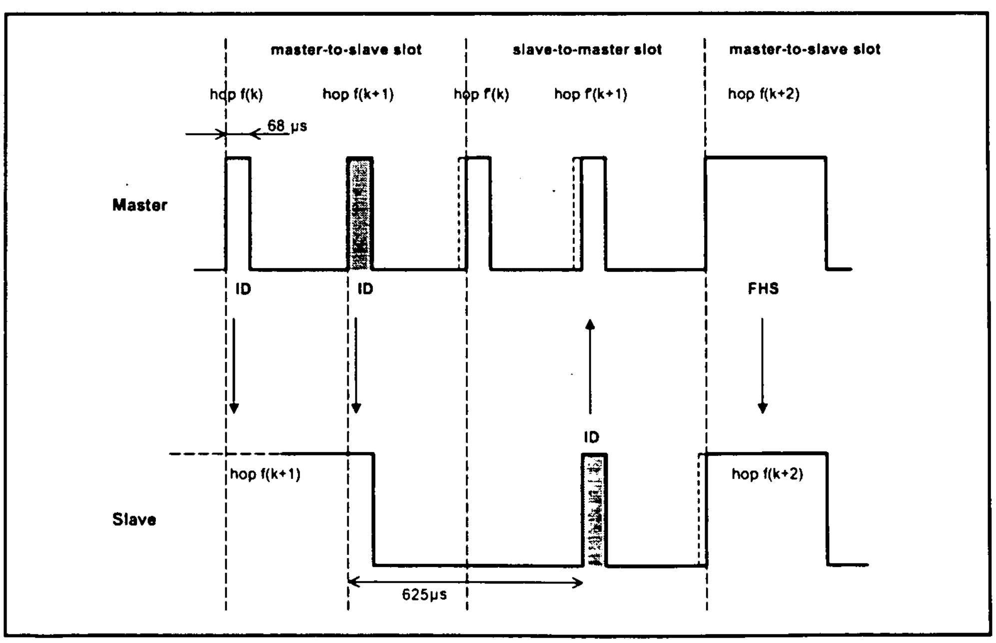


Figure 9.6: Timing of FHS packet on successful page in second half slot.

The slave will adjust its RX/TX timing according to the reception of the **FHS** packet (and not according to the reception of the page message). That is, the second response message that acknowledges the reception of the FHS packet is transmitted 625 μs after the start of the **FHS** packet.

9.7 MULTI-SLAVE OPERATION

As was mentioned in the beginning of this chapter, the master always starts the transmission in the even-numbered slots whereas the slaves start their transmission in the odd-numbered slots. This means that the timing of the master and the slave(s) is shifted by one slot (625 μs), see Figure 9.7 on page 93.

Only the slave that is addressed by its AM_ADDR can return a packet in the next slave-to-master slot. If no valid AM_ADDR is received, the slave may only respond if it concerns its reserved SCO slave-to-master slot. In case of a broadcast message, no slave is allowed to return a packet (an exception is found in the access window for access requests in the park mode, see Section 10.8.4 on page 115).

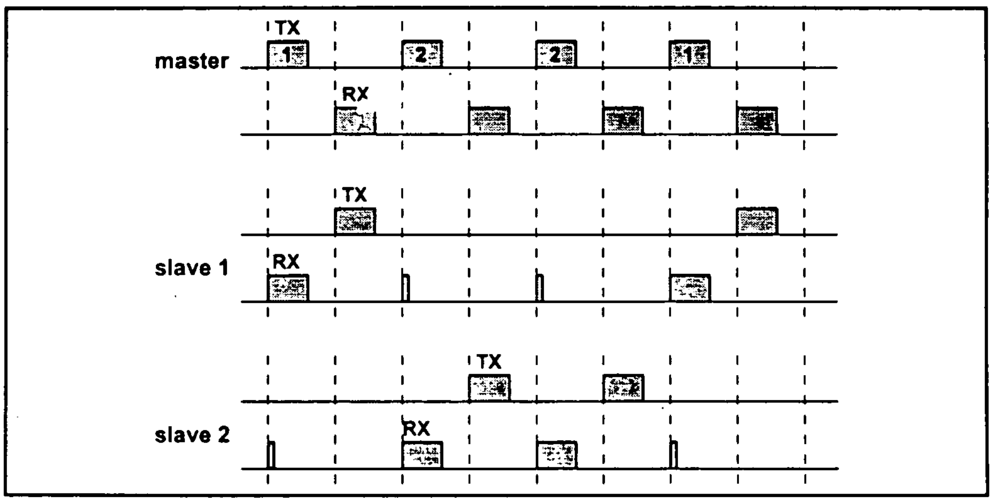


Figure 9.7: RX/TX timing in multi-slave configuration

10 CHANNEL CONTROL

10.1 SCOPE

This section describes how the channel of a piconet is established and how units can be added to and released from the piconet. Several states of operation of the Bluetooth units are defined to support these functions. In addition, the operation of several piconets sharing the same area, the so-called scatternet, is discussed. A special section is attributed to the Bluetooth clock which plays a major role in the FH synchronization.

10.2 MASTER-SLAVE DEFINITION

The channel in the piconet is characterized entirely by the master of the piconet. The Bluetooth device address (BD_ADDR) of the master determines the FH hopping sequence and the channel access code; the system clock of the master determines the phase in the hopping sequence and sets the timing. In addition, the master controls the traffic on the channel by a polling scheme.

By definition, the **master** is represented by the Bluetooth unit that initiates the connection (to one or more **slave** units). Note that the names 'master' and 'slave' only refer to the protocol on the channel: the Bluetooth units themselves are identical; that is, any unit can become a master of a piconet. Once a piconet has been established, master-slave roles can be exchanged. This is described in more detail in Section 10.9.3 on page 123.

10.3 BLUETOOTH CLOCK

Every Bluetooth unit has an internal system clock which determines the timing and hopping of the transceiver. The Bluetooth clock is derived from a free running native clock which is never adjusted and is never turned off. For synchronization with other units, only offsets are used that, added to the native clock, provide temporary Bluetooth clocks which are mutually synchronized. It should be noted that the Bluetooth clock has no relation to the time of day; it can therefore be initialized at any value. The Bluetooth clock provides the heart beat of the Bluetooth transceiver. Its resolution is at least half the TX or RX slot length, or 312.5 μs. The clock has a cycle of about a day. If the clock is implemented with a counter, a 28-bit counter is required that wraps around at 2^{28}-1. The LSB ticks in units of 312.5 μs, giving a clock rate of 3.2 kHz.

The timing and the frequency hopping on the channel of a piconet is determined by the Bluetooth clock of the master. When the piconet is established, the master clock is communicated to the slaves. Each slave adds an offset to its native clock to be synchronized to the master clock. Since the clocks are free-running, the offsets have to be updated regularly.

The clock determines critical periods and triggers the events in the Bluetooth receiver. Four periods are important in the Bluetooth system: 312.5 μs, 625 μs, 1.25 ms, and 1.28 s; these periods correspond to the timer bits CLK_0, CLK_1, CLK_2, and CLK_{12}, respectively, see Figure 10.1 on page 96. Master-to-slave transmission starts at the even-numbered slots when CLK_0 and CLK_1 are both zero.

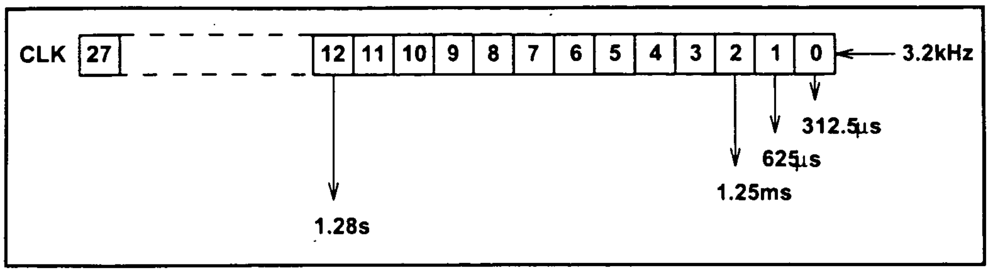


Figure 10.1: Bluetooth clock.

In the different modes and states a Bluetooth unit can reside in, the clock has different appearances:

- CLKN native clock
- CLKE estimated clock
- CLK master clock

CLKN is the free-running native clock and is the reference to all other clock appearances. In states with high activity, the native clock is driven by the reference crystal oscillator with worst case accuracy of +/-20ppm. In the low power states, like **STANDBY, HOLD, PARK**, the native clock may be driven by a low power oscillator (LPO) with relaxed accuracy (+/-250ppm).

CLKE and CLK are derived from the reference CLKN by adding an offset. CLKE is a clock estimate a paging unit makes of the native clock of the recipient; i.e. an offset is added to the CLKN of the pager to approximate the CLKN of the recipient, see Figure 10.2 on page 97. By using the CLKN of the recipient, the pager speeds up the connection establishment.

CLK is the master clock of the piconet. It is used for all timing and scheduling activities in the piconet. All Bluetooth devices use the CLK to schedule their transmission and reception. The CLK is derived from the native clock CLKN by adding an offset, see Figure 10.3 on page 97. The offset is zero for the master since CLK is identical to its own native clock CLKN. Each slave adds an appropriate offset to its CLKN such that the CLK corresponds to the CLKN of the master. Although all CLKNs in the Bluetooth devices run at the same nominal rate, mutual drift causes inaccuracies in CLK. Therefore, the offsets in the slaves must be regularly updated such that CLK is approximately CLKN of the master.

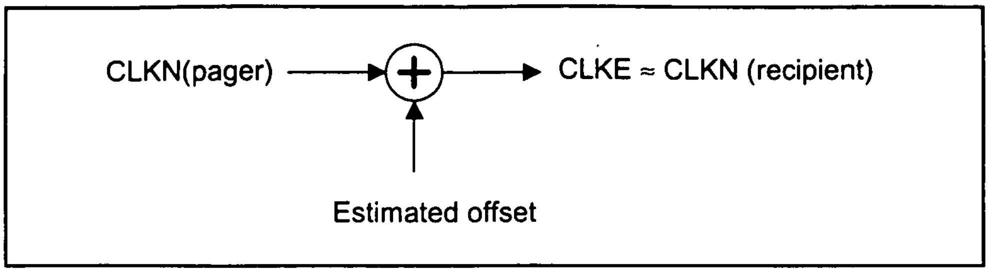


Figure 10.2: Derivation of CLKE

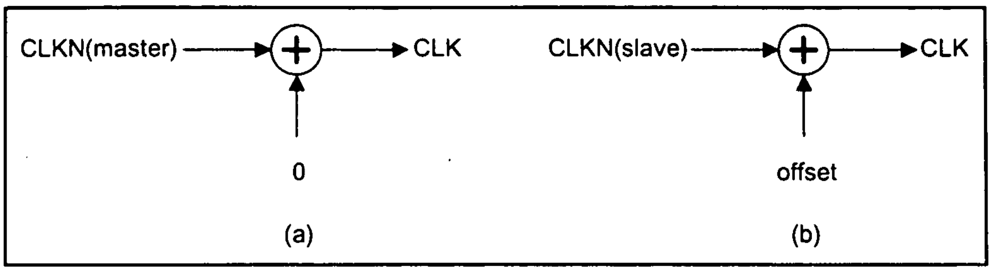
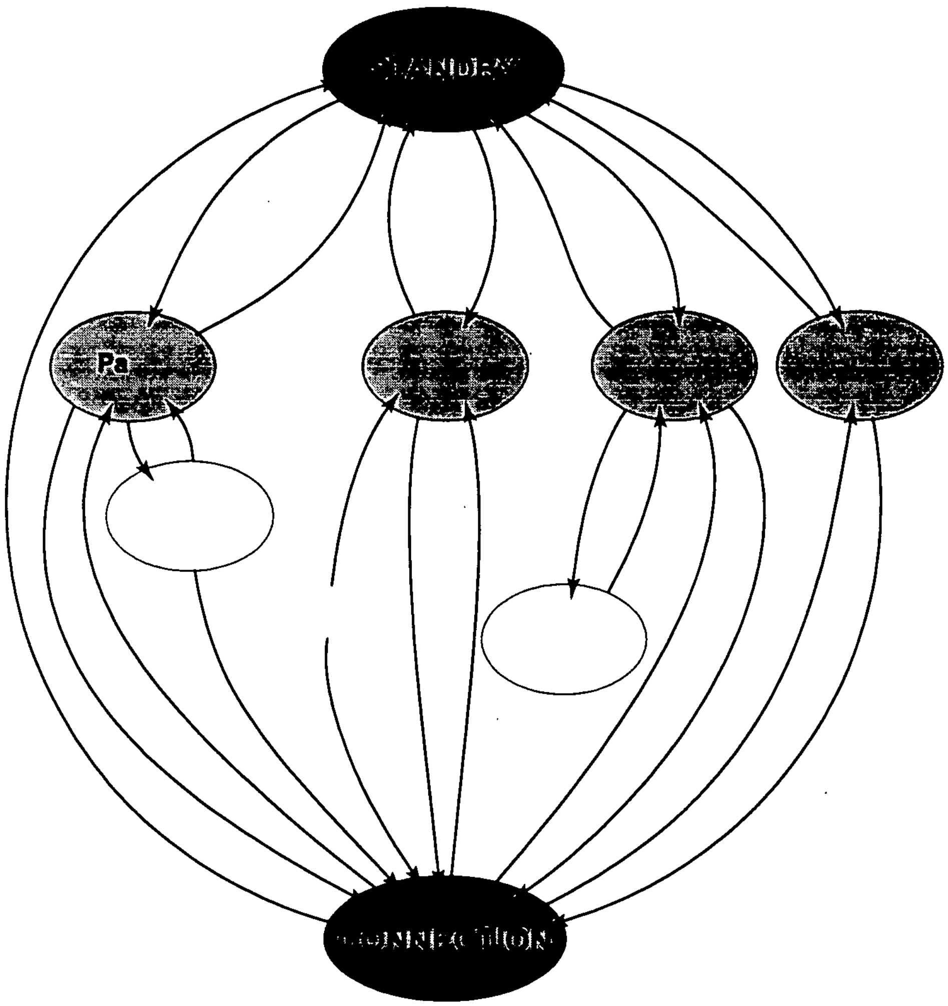


Figure 10.3: Derivation of CLK in master (a) and in slave (b).

10.4 OVERVIEW OF STATES

Figure 10.4 on page 98 shows a state diagram illustrating the different states used in the Bluetooth link controller. There are two major states: **STANDBY** and **CONNECTION**; in addition, there are seven substates, **page**, **page scan**, **inquiry**, **inquiry scan**, **master response**, **slave response**, and **inquiry response**. The substates are interim states that are used to add new slaves to a piconet. To move from one state to the other, either commands from the Bluetooth link manager are used, or internal signals in the link controller are used (such as the trigger signal from the correlator and the timeout signals).



10.5 STANDBY STATE

The **STANDBY** state is the default state in the Bluetooth unit. In this state, the Bluetooth unit is in a low-power mode. Only the native clock is running at the accuracy of the LPO (or better).

The controller may leave the **STANDBY** state to scan for page or inquiry messages, or to page or inquiry itself. When responding to a page message, the unit will not return to the **STANDBY** state but enter the **CONNECTION** state as a slave. When carrying out a successful page attempt, the unit will enter the **CONNECTION** state as a master. The intervals with which scan activities can be carried out are discussed in and

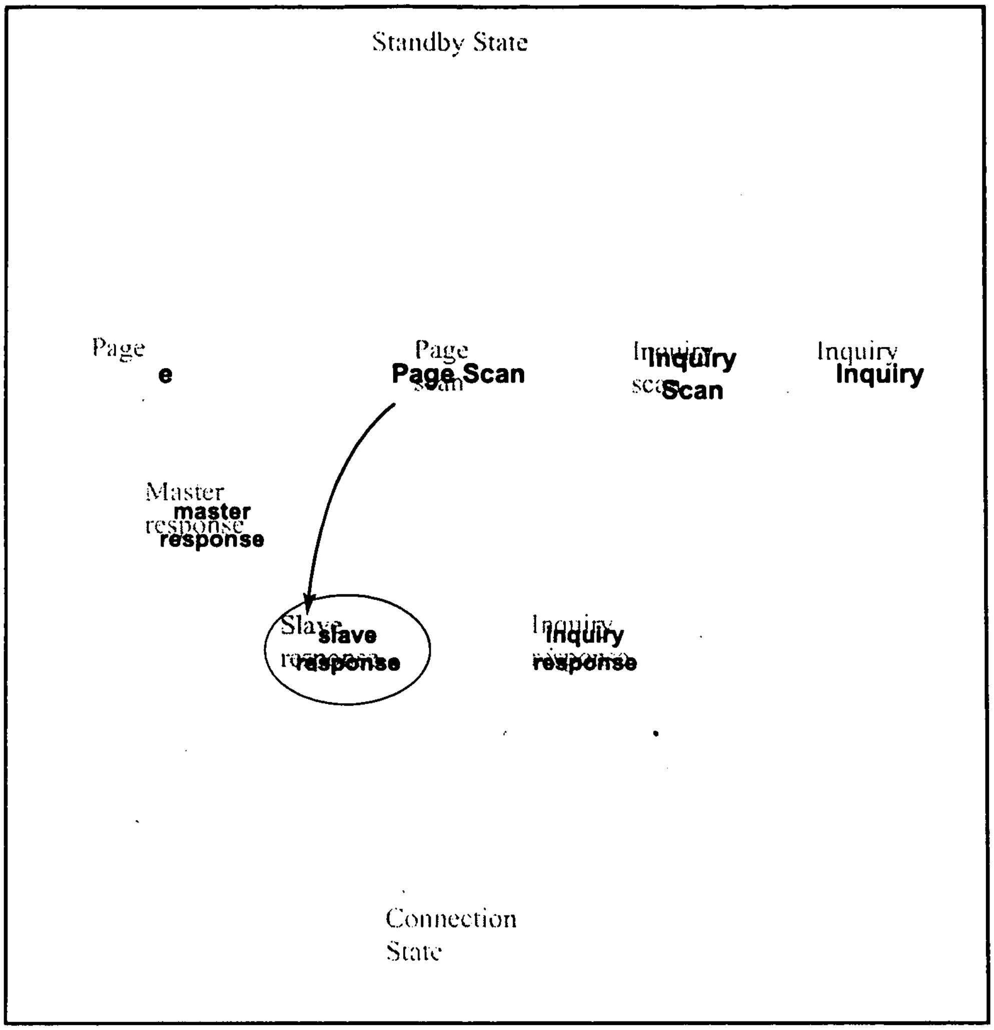


Figure 10.4: State diagram of Bluetooth link controller.

Section 10.6.2 on page 99 Section 10.7.2

on page 109

10.6 ACCESS PROCEDURES

10.6.1 General

In order to establish new connections the procedures inquiry and paging are used. The inquiry procedure enables a unit to discover which units are in range, and what their device addresses and clocks are. With the paging procedure, an actual connection can be established. Only the Bluetooth device address is required to set up a connection. Knowledge about the clock will accelerate the setup procedure. A unit that establishes a connection will carry out a page procedure and will automatically be the master of the connection.

In the paging and inquiry procedures, the device access code (DAC) and the inquiry access code (IAC) are used, respectively. A unit in the **page scan** or **inquiry scan** substate correlates against these respective access codes with a matching correlator.

For the paging process, several paging schemes can be applied. There is one mandatory paging scheme which has to be supported by each Bluetooth device. This mandatory scheme is used when units meet for the first time, and in case the paging process directly follows the inquiry process. Two units, once connected using a mandatory paging/scanning scheme, may agree on an optional paging/scanning scheme. Optional paging schemes are discussed in "Appendix VII" on page 999. In the current chapter, only the mandatory paging scheme is considered.

10.6.2 Page scan

In the **page scan** substate, a unit listens for its own device access code for the duration of the scan window $T_{w\ page\ scan}$. During the scan window, the unit listens at a single hop frequency, its correlator matched to its device access code. The scan window shall be long enough to completely scan 16 page frequencies.

When a unit enters the **page scan** substate, it selects the scan frequency according to the page hopping sequence corresponding to this unit, see Section 11.3.1 on page 135. This is a 32-hop sequence (or a 16-hop sequence in case of a reduced-hop system) in which each hop frequency is unique. The page hopping sequence is determined by the unit's Bluetooth device address (BD_ADDR). The phase in the sequence is determined by $CLKN_{16\text{-}12}$ of the unit's native clock ($CLKN_{15\text{-}12}$ in case of a reduced-hop system); that is, every 1.28s a different frequency is selected.

If the correlator exceeds the trigger threshold during the **page scan**, the unit will enter the **slave response** substate, which is described in Section 10.6.4.1 on page 105.

The **page scan** substate can be entered from the **STANDBY** state or the **CONNECTION** state. In the **STANDBY** state, no connection has been established and the unit can use all the capacity to carry out the **page scan**. Before entering the **page scan** substate from the **CONNECTION** state, the unit preferably reserves as much capacity for scanning. If desired, the unit may place ACL connections in the HOLD mode or even use the PARK mode, see Section 10.8.3 on page 114 and Section 10.8.4 on page 115. SCO connections are preferably not interrupted by the **page scan**. In this case, the **page scan** may be interrupted by the reserved SCO slots which have higher priority than the **page scan**. SCO packets should be used requiring the least amount of capacity (**HV3** packets). The scan window shall be increased to minimize the setup delay. If one SCO link is present using **HV3** packets and T_{SCO}=6 slots, a total scan window $T_{w\ page\ scan}$ of at least 36 slots (22.5ms) is recommended; if two SCO links are present using **HV3** packets and T_{SCO}=6 slots, a total scan window of at least 54 slots (33.75ms) is recommended.

The scan interval $T_{page\ scan}$ is defined as the interval between the beginnings of two consecutive page scans. A distinction is made between the case where the scan interval is equal to the scan window $T_{w\ page\ scan}$ (continuous scan), the scan interval is maximal 1.28s, or the scan interval is maximal 2.56s. These three cases determine the behavior of the paging unit; that is, whether the paging unit shall use R0, R1 or R2, see also Section 10.6.3 on page 101.
Table 10.1 illustrates the relationship between $T_{page\ scan}$ and modes R0, R1 and R2. Although scanning in the R0 mode is continuous, the scanning may be interrupted by for example reserved SCO slots. The scan interval information is included in the SR field in the FHS packet.

During page scan the Bluetooth unit may choose to use an optional scanning scheme. (An exception is the page scan after returning an inquiry response message. See Section 10.7.4 on page 111 for details.)

SR mode	$T_{page scan}$	N_{page}
R0	continuous	≥1
R1	≤ 1.28s	≥ 128
R2	≤ 2.56s	≥ 256
Reserved	-	-

Table 10.1: Relationship between scan interval, train repetition, and paging modes R0, R1 and R2.

10.6.3 Page

The **page** substate is used by the master (source) to activate and connect to a slave (destination) which periodically wakes up in the **page scan** substate. The master tries to capture the slave by repeatedly transmitting the slave's device access code (DAC) in different hop channels. Since the Bluetooth clocks of the master and the slave are not synchronized, the master does not know exactly when the slave wakes up and on which hop frequency. Therefore, it transmits a train of identical DACs at different hop frequencies, and listens in between the transmit intervals until it receives a response from the slave.

The page procedure in the master consists of a number of steps. First, the slave's device address is used to determine the page hopping sequence, see Section 11.3.2 on page 135. This is the sequence the master will use to reach the slave. For the phase in the sequence, the master uses an estimate of the slave's clock. This estimate can for example be derived from timing information that was exchanged during the last encounter with this particular device (which could have acted as a master at that time), or from an inquiry procedure. With this estimate CLKE of the slave's Bluetooth clock, the master can predict when the slave wakes up and on which hop channel.

The estimate of the Bluetooth clock in the slave can be completely wrong. Although the master and the slave use the same hopping sequence, they use different phases in the sequence and will never meet each other. To compensate for the clock drifts, the master will send its page message during a short time interval on a number of wake-up frequencies. It will in fact transmit also on hop frequencies just before and after the current, predicted hop frequency. During each TX slot, the master sequentially transmits on two different hop frequencies. Since the page message is the ID packet which is only 68 bits in length, there is ample of time (224.5 μs minimal) to switch the synthesizer. In the following RX slot, the receiver will listen sequentially to two corresponding RX hops for ID packet. The RX hops are selected according to the page_response hopping sequence. The page_response hopping sequence is strictly related to the page hopping sequence; that is: for each page hop there is a corresponding page_response hop. The RX/TX timing in the **page** sub-

state has been described in Section 9, see also Figure 9.4 on page 91. In the next TX slot, it will transmit on two hop frequencies different from the former ones. The synthesizer hop rate is increased to 3200 hops/s.

A distinction must be made between the 79-hop systems and the 23-hop systems. First the 79-hop systems are considered. With the increased hopping rate as described above, the transmitter can cover 16 different hop frequencies in 16 slots or 10 ms. The page hopping sequence is divided over two paging trains **A** and **B** of 16 frequencies. Train **A** includes the 16 hop frequencies surrounding the current, predicted hop frequency f(k), where k is determined by the clock estimate $CLKE_{16-12}$. So the first train consists of hops

f(k-8), f(k-7),...,f(k),...,f(k+7)

When the difference between the Bluetooth clocks of the master and the slave is between -8x1.28 s and +7x1.28 s, one of the frequencies used by the master will be the hop frequency the slave will listen to. However, since the master does not know when the slave will enter the **page scan** substate, he has to repeat this train **A** N_{page} times or until a response is obtained. If the slave scan interval corresponds to R1, the repetition number is at least 128; if the slave scan interval corresponds to R2, the repetition number is at least 256.
Note that $CLKE_{16-12}$ changes every 1.28 s; therefore, every 1.28 s, the trains will include different frequencies of the page hopping set.

When the difference between the Bluetooth clocks of the master and the slave is less than -8x1.28 s or larger than +7x1.28 s, more distant hops must be probed. Since in total, there are only 32 dedicated wake-up hops, the more distant hops are the remaining hops not being probed yet. The remaining 16 hops are used to form the new 10 ms train **B.** The second train consists of hops

f(k-16), f(k-15),...,f(k-9),f(k+8)...,f(k+15)

Train **B** is repeated for N_{page} times. If still no response is obtained, the first train **A** is tried again N_{page} times. Alternate use of train A and train B is continued until a response is received or the timeout *pageTO* is exceeded. If during one of the listening occasions, a response is returned by the slave, the master unit enters the **master response** substate.

The description for paging and **page scan** procedures given here has been tailored towards the 79-hop systems used in the US and Europe. For the 23-hop systems as used in Japan and some European countries, the procedure is slightly different. In the 23-hop case, the length of the page hopping sequence is reduced to 16. As a consequence, there is only a single train (train **A**) including all the page hopping frequencies. The phase to the page hopping sequence is not $CLKE_{16-12}$ but $CLKE_{15-12}$. An estimate of the slave's clock does not have to be made.

The **page** substate can be entered from the **STANDBY** state or the **CONNECTION** state. In the **STANDBY** state, no connection has been established and

the unit can use all the capacity to carry out the page. Before entering the page substate from the CONNECTION state, the unit shall free as much capacity as possible for scanning. To ensure this, it is recommended that the ACL connections are put on hold or park. However, the SCO connections shall not be disturbed by the page. This means that the page will be interrupted by the reserved SCO slots which have higher priority than the page. In order to obtain as much capacity for paging, it is recommended to use the SCO packets which use the least amount of capacity (**HV3** packets). If SCO links are present, the repetition number N_{page} of a single train shall be increased, see Table 10.2. Here it has been assumed that the **HV3** packet are used with an interval T_{SCO}=6 slots, which would correspond to a 64 kb/s voice link.

SR mode	**no SCO link**	**one SCO link (HV3)**	**two SCO links (HV3)**
R0	$N_{page} \geq 1$	$N_{page} \geq 2$	$N_{page} \geq 3$
R1	$N_{page} \geq 128$	$N_{page} \geq 256$	$N_{page} \geq 384$
R2	$N_{page} \geq 256$	$N_{page} \geq 512$	$N_{page} \geq 768$

Table 10.2: Relationship between train repetition, and paging modes R0, R1 and R2 when SCO links are present.

The construction of the page train is independent on the presence of SCO links; that is, SCO packets are sent on the reserved slots but do not affect the hop frequencies used in the unreserved slots, see Figure 10.5 on page 103.

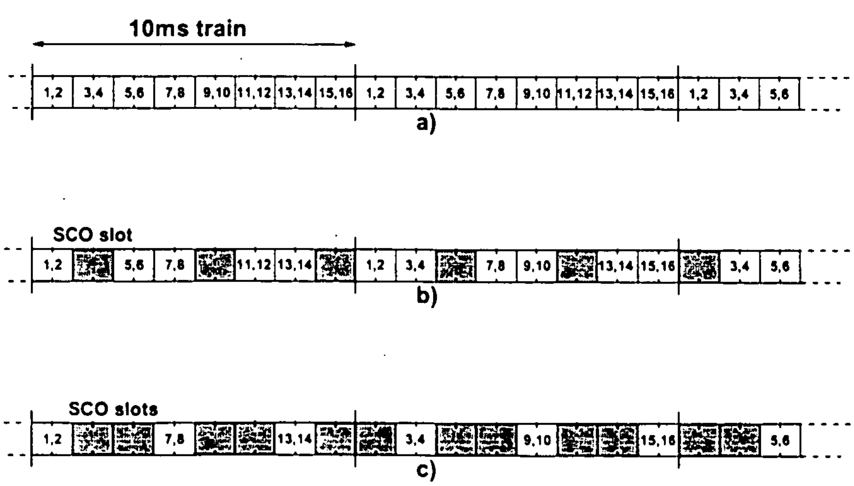


Figure 10.5: Conventional page (a), page while one SCO link present (b), page while two SCO links present (c).

For the descriptions of optional paging schemes see "Appendix VII" on page 999.

10.6.4 Pag response procedures

When a page message is successfully received by the slave, there is a coarse FH synchronization between the master and the slave. Both the master and the slave enter a response routine to exchange vital information to continue the connection setup. Important for the piconet connection is that both Bluetooth units use the same channel access code, use the same channel hopping sequence, and that their clocks are synchronized. These parameters are derived from the master unit. The unit that initializes the connection (starts paging) is defined as the master unit (which is thus only valid during the time the piconet exists). The channel access code and channel hopping sequence are derived from the Bluetooth device address (BD_ADDR) of the master. The timing is determined by the master clock. An offset is added to the slave's native clock to temporarily synchronize the slave clock to the master clock. At start-up, the master parameters have to be transmitted from the master to the slave. The messaging between the master and the slave at start-up will be considered in this section.

The initial messaging between master and slave is shown in Table 10.3 on page 104 and in Figure 10.6 on page 105 and Figure 10.7 on page 105. In those two figures frequencies f (k), f(k+1), etc. are the frequencies of the page hopping sequence determined by the slave's BD_ADDR. The frequencies f'(k), f'(k+1), etc. are the corresponding page_response frequencies (slave-to-master). The frequencies g(m) belong to the channel hopping sequence.

Step	Message	Direction	Hopping Sequence	Access Code and Clock
1	slave ID	master to slave	page	slave
2	slave ID	slave to master	page response	slave
3	FHS	master to slave	page	slave
4	slave ID	slave to master	page response	slave
5	1st packet master	master to slave	channel	master
6	1st packet slave	slave to master	channel	master

Table 10.3: Initial messaging during start-up.

In step 1 (see Table 10.3 on page 104), the master unit is in **page** substate and the slave unit in the **page scan** substate. Assume in this step that the page message (= slave's device access code) sent by the master reaches the slave. On recognizing its device access code, the slave enters the **slave response** in step 2. The master waits for a reply from the slave and when this arrives in step 2, it will enter the **master response** in step 3. Note that during the initial message exchange, all parameters are derived from the slave's BD_ADDR, and that only the page hopping and page_response hopping sequences are used

(which are also derived from the slave's BD_ADDR). Note that when the master and slave enter the response states, their clock input to the page and page_response hop selection is frozen as is described in Section 11.3.3 on page 136.

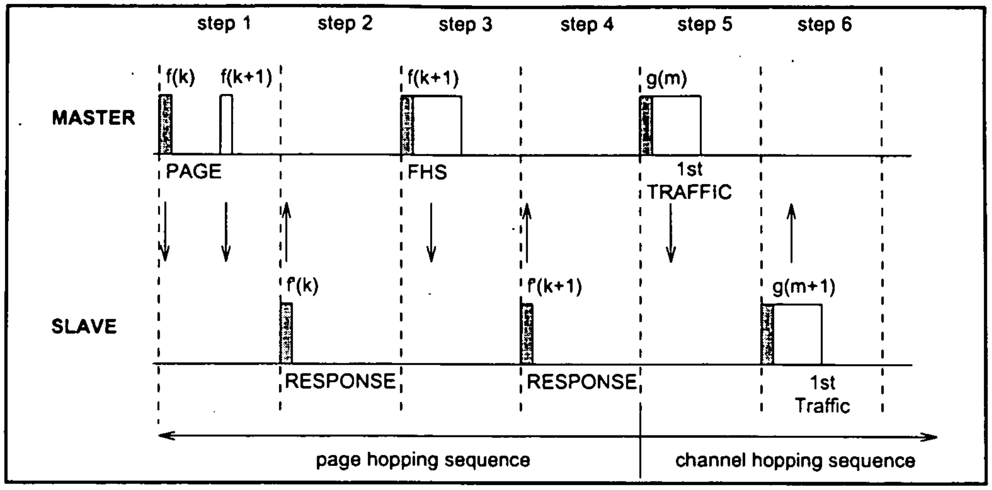


Figure 10.6: Messaging at initial connection when slave responds to first page message.

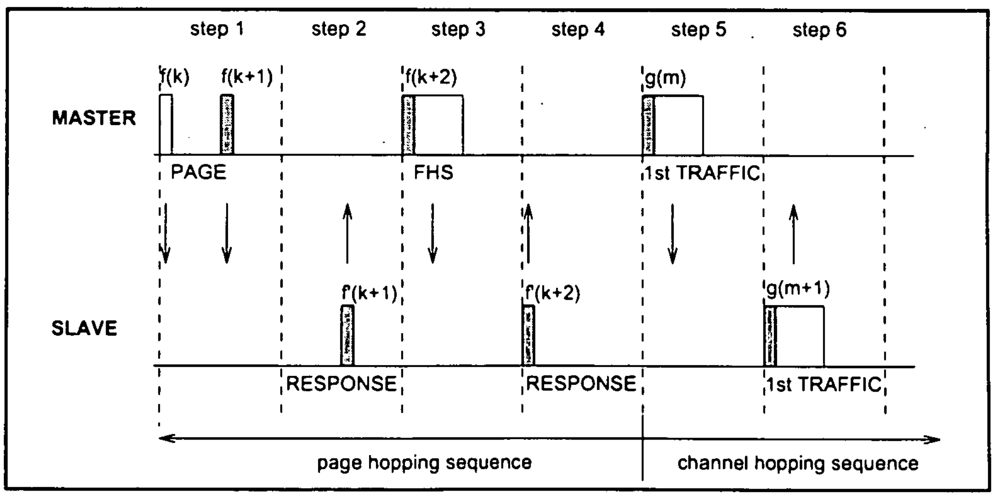


Figure 10.7: Messaging at initial connection when slave responds to second page message.

10.6.4.1 Slave response

After having received its own device access code in step 1, the slave unit transmits a response message in step 2. This response message again only consists of the slave's device access code. The slave will transmit this response 625 μs after the beginning of the received page message (slave ID packet) and at the response hop frequency that corresponds to the hop frequency in which the page message was received. The slave transmission is therefore time

aligned to the master transmission. During initial messaging, the slave still uses the page response hopping sequence to return information to the master. The clock input $CLKN_{16-12}$ is frozen at the value it had at the time the page message was received.

After having sent the response message, the slave's receiver is activated (312.5 μs after the start of the response message) and awaits the arrival of a **FHS** packet. Note that a **FHS** packet can already arrive 312.5 μs after the arrival of the page message as shown in Figure 10.7 on page 105, and not after 625 μs as is usually the case in the RX/TX timing. More details about the timing can be found in Section 9.6 on page 91.

If the setup fails before the **CONNECTION** state has been reached, the following procedure is carried out. The slave will keep listening as long as no **FHS** packet is received until *pagerespTO* is exceeded. Every 1.25 ms, however, it will select the next master-to-slave hop frequency according to the page hop sequence. If nothing is received after *pagerespTO*, the slave returns back to the **page scan** substate for one scan period. Length of the scan period depends on the SCO slots present. If no page message is received during this additional scan period, the slave will resume scanning at its regular scan interval and return to the state it was in prior to the first page scan state.

If a **FHS** packet is received by the slave in the **slave response** substate, the slave returns a response (slave's device access code only) in step 4 to acknowledge the reception of the **FHS** packet (still using the page response hopping sequence). The transmission of this response packet is based on the reception of the **FHS** packet. Then the slave changes to the channel (master's) access code and clock as received from the **FHS** packet. Only the 26 MSBs of the master clock are transferred: the timing is assumed such that CLK_1 and CLK_0 are both zero at the time the **FHS** packet was received as the master transmits in even slots only. From the master clock in the **FHS** packet, the offset between the master's clock and the slave's clock is determined and reported to the slave's link manager.

Finally, the slave enters the **CONNECTION** state in step 5. From then on, the slave will use the master's clock and the master BD_ADDR to determine the channel hopping sequence and the channel access code. The connection mode starts with a POLL packet transmitted by the master. The slave responds with any type of packet. If the POLL packet is not received by the slave, or the response packet is not received by the master, within *newconnectionTO* number of slots after FHS packet acknowledgement, the master and the slave will return to page and page scan substates, respectively.See Section 10.8 on page 112

10.6.4.2 Master response

When the master has received a response message from the slave in step 2, it will enter the **master response** routine. It freezes the current clock input to the page hop selection scheme. Then the master will transmit a **FHS** packet in step 3 containing the master's real-time Bluetooth clock, the master's 48-bit BD_ADDR address, the BCH parity bits, and the class of device. The **FHS** packet contains all information to construct the channel access code without requiring a mathematical derivation from the master device address. The **FHS** packet is transmitted at the beginning of the master-to-slave slot following the slot in which the slave has responded. So the TX timing of the **FHS** is not based on the reception of the response packet from the slave. The **FHS** packet may therefore be sent 312.5 µs after the reception of the response packet like shown in Figure 10.7 on page 105 and not 625 µs after the received packet as is usual in the RX/TX timing, see also Section 9.6 on page 91.

After the master has sent its **FHS** packet, it waits for a second response from the slave in step 4 which acknowledges the reception of the **FHS** packet. Again this is only the slave's device access code. If no response is received, the master retransmits the **FHS** packet, but with an updated clock and still using the slave's parameters. It will retransmit (the clock is updated every retransmission) until a second slave response is received, or the timeout of *pagerespTO* is exceeded. In the latter case, the master turns back to the **page** substate and sends an error message to the link manager. During the retransmissions of the **FHS** packet, the master keeps using the page hopping sequence.

If the slave's response is indeed received, the master changes to the master parameters, so the channel access code and the master clock. The lower clock bits CLK_0 and CLK_1 are zero at the start of the **FHS** packet transmission and are not included in the **FHS** packet. Finally, the master enters the **CONNECTION** state in step 5. The master BD_ADDR is used to change to a new hopping sequence, the *channel hopping sequence*. The channel hopping sequence uses all 79 hop channels in a (pseudo) random fashion, see also Section 11.3.6 on page 138. The master can now send its first traffic packet in a hop determined with the new (master) parameters. This first packet will be a POLL packet. See Section 10.8 on page 112.

The master can now send its first traffic packet in a hop determined with the new (master) parameters. The first packet in this state is a POLL packet sent by the master. This packet will be sent within *newconnectionTO* number of slots after reception of the FHS packet acknowledgement. The slave will respond with any type of packet. If the POLL packet is not received by the slave or the POLL packet response is not received by the master within *newconnectionTO* number of slots, the master and the slave will return to page and page scan substates, respectively.

10.7 INQUIRY PROCEDURES

10.7.1 General

In the Bluetooth system, an inquiry procedure is defined which is used in applications where the destination's device address is unknown to the source. One can think of public facilities like printers or facsimile machines, or access points to a LAN. Alternatively, the inquiry procedure can be used to discover which other Bluetooth units are within range. During an **inquiry** substate, the discovering unit collects the Bluetooth device addresses and clocks of all units that respond to the inquiry message. It can then, if desired, make a connection to any one of them by means of the previously described page procedure.

The inquiry message broadcasted by the source does not contain any information about the source. However, it may indicate which class of devices should respond. There is one general inquiry access code (GIAC) to inquire for any Bluetooth device, and a number of dedicated inquiry access codes (DIAC) that only inquire for a certain type of devices. The inquiry access codes are derived from reserved Bluetooth device addresses and are further described in Section 4.2.1.

A unit that wants to discover other Bluetooth units enters an **inquiry** substate. In this substate, it continuously transmits the inquiry message (which is the ID packet, see Section 4.4.1.1 on page 55) at different hop frequencies. The **inquiry** hop sequence is always derived from the LAP of the GIAC. Thus, even when DIACs are used, the applied hopping sequence is generated from the GIAC LAP. A unit that allows itself to be discovered, regularly enters the **inquiry scan** substate to respond to inquiry messages. The following sections describe the message exchange and contention resolution during inquiry response. The inquiry response is optional: a unit is not forced to respond to an inquiry message.

10.7.2 Inquiry scan

The **inquiry scan** substate is very similar to the **page scan** substate. However, instead of scanning for the unit's device access code, the receiver scans for the inquiry access code long enough to completely scan for 16 inquiry frequencies. The length of this scan period is denoted $T_{w\_inquiry\_scan}$. The scan is performed at a single hop frequency. As in the page procedure, the inquiry procedure uses 32 dedicated inquiry hop frequencies according to the *inquiry hopping sequence*. These frequencies are determined by the general inquiry address. The phase is determined by the native clock of the unit carrying out the **inquiry scan**; the phase changes every 1.28s.

Instead or in addition to the general inquiry access code, the unit may scan for one or more dedicated inquiry access codes. However, the scanning will follow the inquiry hopping sequence which is determined by the general inquiry address. If an inquiry message is recognized during an inquiry wake-up period, the Bluetooth unit enters the **inquiry response** substate.

The **inquiry scan** substate can be entered from the **STANDBY** state or the **CONNECTION** state. In the **STANDBY** state, no connection has been established and the unit can use all the capacity to carry out the **inquiry scan**. Before entering the **inquiry scan** substate from the **CONNECTION** state, the unit preferably reserves as much capacity as possible for scanning. If desired, the unit may place ACL connections in the HOLD mode or even use the PARK mode, see Section 10.8.3 on page 114. SCO connections are preferably not interrupted by the **inquiry scan**. In this case, the **inquiry scan** may be interrupted by the reserved SCO slots which have higher priority than the **inquiry scan**. SCO packets should be used requiring the least amount of capacity (**HV3** packets). The scan window, $T_{w\ inquiry\ scan}$, shall be increased to increase the probability to respond to an inquiry message. If one SCO link is present using HV3 packets and T_{SCO}=6 slots, a total scan window of at least 36 slots (22.5ms) is recommended; if two SCO links are present using HV3 packets and T_{SCO}=6 slots, a total scan window of at least 54 slots (33.75ms) is recommended.

The scan interval $T_{inquiry\ scan}$ is defined as the interval between two consecutive inquiry scans. The **inquiry scan** interval shall be at most 2.56 s.

10.7.3 Inquiry

The **inquiry** substate is used by the unit that wants to discover new devices. This substate is very similar to the **page** substate, the same TX/RX timing is used as used for paging, see Section 9.6 on page 91 and Figure 9.4 on page 91. The TX and RX frequencies follow the inquiry hopping sequence and the inquiry response hopping sequence, and are determined by the general inquiry access code and the native clock of the discovering device. In between inquiry transmissions, the Bluetooth receiver scans for inquiry response messages. When found, the entire response packet (which is in fact a **FHS** packet) is read, after which the unit continues with the inquiry transmissions. So the Bluetooth unit in an **inquiry** substate does not acknowledge the inquiry response messages. It keeps probing at different hop channels and in between listens for response packets. Like in the **page** substate, two 10 ms trains **A** and **B** are defined, splitting the 32 frequencies of the inquiry hopping sequence into two 16-hop parts. A single train must be repeated for at least $N_{inquiry}$=256 times before a new train is used. In order to collect all responses in an error-free environment, at least three train switches must have taken place. As a result, the **inquiry** substate may have to last for 10.24 s unless the inquirer collects enough responses and determines to abort the inquiry substate earlier. If desired, the inquirer can also prolong the inquiry substate to increase the probability of receiving all responses in an error-prone environment. If an inquiry procedure is automatically initiated periodically (say a 10 s period every minute), then the interval between two inquiry instances must be determined randomly. This is done to avoid two Bluetooth units to synchronize their inquiry procedures.

The **inquiry** substate is continued until stopped by the Bluetooth link manager (when it decides that it has sufficient number of responses), or when a timeout has been reached (*inquiryTO*).

The **inquiry** substate can be entered from the **STANDBY** state or the **CONNECTION** state. In the **STANDBY** state, no connection has been established and the unit can use all the capacity to carry out the inquiry. Before entering the inquiry substate from the **CONNECTION** state, the unit shall free as much capacity as possible for scanning. To ensure this, it is recommended that the ACL connections are put on hold or park. However, the SCO connections shall not be disturbed by the inquiry. This means that the inquiry will be interrupted by the reserved SCO slots which have higher priority than the inquiry. In order to obtain as much capacity for inquiry, it is recommended to use the SCO packets which use the least amount of capacity (**HV3** packets). If SCO links are present, the repetition number $N_{inquiry}$ shall be increased, see Table 10.4 on page 111.
Here it has been assumed that the **HV3** packet are used with an interval T_{SCO}=6 slots, which would correspond to a 64 kb/s voice link.

	one SCO link	one SCO link (HV3)	two SCO links (HV3)
$N_{inquiry}$	≥ 256	≥ 512	≥ 768

Table 10.4: Increase of train repetition when SCO links are present.

10.7.4 Inquiry response

For the inquiry operation, there is only a slave response, no master response. The master listens between inquiry messages for responses, but after reading a response, it continues to transmit inquiry messages. The slave response routine for inquiries differs completely from the slave response routine applied for pages. When the inquiry message is received in the **inquiry scan** substate, a response message containing the recipient's address must be returned. This response message is a conventional **FHS** packet carrying the unit's parameters. However, a contention problem may arise when several Bluetooth units are in close proximity to the inquiring unit and all respond to an inquiry message at the same time. First of all, every Bluetooth unit has a free running clock; therefore, it is highly unlikely that they all use the same phase of the inquiry hopping sequence. However, in order to avoid collisions between units that do wake up in the same inquiry hop channel simultaneously, the following protocol in the slave's **inquiry response** is used. If the slave receives an inquiry message, it generates a random number RAND between 0 and 1023. In addition, it freezes the current input value (phase) to the hop selection scheme, see also Section 11.3.5 on page 137. The slave then returns to the **CONNECTION** or **STANDBY** state for the duration of RAND time slots. Before returning to the **CONNECTION** or **STANDBY** state, the unit may go through the page scan substate; this page scan must use the mandatory page scan scheme. After at least RAND slots, the unit will return to the **inquiry response** substate. On the first inquiry message received the slave returns an **FHS** response packet to the master. If during the scan no trigger occurs within a timeout period of *inqrespTO*, the slave returns to the **STANDBY** or **CONNECTION** state. If the unit does receive an inquiry message and returns an FHS packet, it adds an offset of 1 to the phase in the inquiry hop sequence (the phase has a 1.28 s resolution) and enters the **inquiry scan** substate again. If the slave is triggered again, it repeats the procedure using a new RAND. The offset to the clock accumulates each time a **FHS** packet is returned. During a 1.28 s probing window, a slave on average responses 4 times, but on different frequencies and at different times. Possible SCO slots should have priority over response packets; that is, if a response packet overlaps with an SCO slot, it is not sent but the next inquiry message is awaited.

The messaging during the inquiry routines is summarized in Table 10.5 on page 112. In step 1, the master transmits an inquiry message using the inquiry access code and its own clock. The slave responds with the **FHS** packet which contains the slave's device address, native clock and other slave information. This **FHS** packet is returned at a semi-random time. The **FHS** packet is not acknowledged in the inquiry routine, but it is retransmitted at other times and frequencies as long as the master is probing with inquiry messages.

step	message	direction	hopping sequence	access code
1	ID	master to slave	inquiry	inquiry
2	FHS	slave to master	inquiry response	inquiry

Table 10.5: Messaging during inquiry routines.

If the scanning unit uses an optional scanning scheme, after responding to an inquiry with an FHS packet, it will perform page scan using the mandatory page scan scheme for $T_{mandatory\ pscan}$ period. Every time an inquiry response is sent the unit will start a timer with a timeout of $T_{mandatory\ pscan}$. The timer will be reset at each new inquiry response. Until the timer times out, when the unit performs page scan, it will use the mandatory page scanning scheme in the SR mode it uses for all its page scan intervals. Using the mandatory page scan scheme after the inquiry procedure enables all units to connect even if they do not support an optional paging scheme (yet). In addition to using the mandatory page scan scheme, an optional page scan scheme can be used in parallel for the $T_{mandatory\ pscan}$ period.

The $T_{mandatory\ pscan}$ period is included in the SP field of the FHS packet returned in the inquiry response routine, see Section 4.4.1.4 on page 56. The value of the period is indicated in the Table 10.6

SP mode	$T_{mandatory\ pscan}$
P0	≥20s
P1	≥ 40s
P2	≥ 60s
Reserved	-

Table 10.6: Mandatory scan periods for P0, P1, P2 scan period modes.

10.8 CONNECTION STATE

In the **CONNECTION** state, the connection has been established and packets can be sent back and forth. In both units, the channel (master) access code and the master Bluetooth clock are used. The hopping scheme uses the *channel hopping sequence.* The master starts its transmission in even slots (CLK_{1-0}=00), the slave starts its transmission in odd slots (CLK_{1-0}=10)

The **CONNECTION** state starts with a POLL packet sent by the master to verify the switch to the master's timing and channel frequency hopping. The slave can respond with any type of packet. If the slave does not receive the POLL packet or the master does not receive the response packet for *newconnectionTO* number of slots, both devices will return to **page/page scan** substates.

The first information packets in the **CONNECTION** state contain control messages that characterize the link and give more details regarding the Bluetooth units. These messages are exchanged between the link managers of the units. For example, it defines the SCO links and the sniff parameters. Then the transfer of user information can start by alternately transmitting and receiving packets.

The **CONNECTION** state is left through a **detach** or **reset** command. The **detach** command is used if the link has been disconnected in the normal way. All configuration data in the Bluetooth link controller is still valid. The **reset** command is a hard reset of all controller processes. After a reset, the controller has to be reconfigured.

The Bluetooth units can be in several modes of operation during the **CONNECTION** state: active mode, sniff mode, hold mode, and park mode. These modes are now described in more detail.

10.8.1 Active mode

In the active mode, the Bluetooth unit actively participates on the channel. The master schedules the transmission based on traffic demands to and from the different slaves. In addition, it supports regular transmissions to keep slaves synchronized to the channel. Active slaves listen in the master-to-slave slots for packets. If an active slave is not addressed, it may sleep until the next new master transmission. From the type indication in the packet, the number of slots the master has reserved for its transmission can be derived; during this time, the non-addressed slaves do not have to listen on the master-to-slave slots. A periodic master transmission is required to keep the slaves synchronized to the channel. Since the slaves only need the channel access code to synchronize with, any packet type can be used for this purpose.

10.8.2 Sniff mod

In the sniff mode, the duty cycle of the slave's listen activity can be reduced. If a slave participates on an ACL link, it has to listen in every ACL slot to the master traffic. With the sniff mode, the time slots where the master can start transmission to a specific slave is reduced; that is, the master can only start transmission in specified time slots. These so-called sniff slots are spaced regularly with an interval of $T_{sniff.}$

The slave has to listen at D_{sniff} slot every sniff period, T_{sniff} for a $N_{sniff\ attempt}$ number of times. If the slave receives a packet in one of the $N_{sniff\ attempt}$ RX slots, it should continue listening as long as it receives packets to its own AM_ADDR. Once it stops receiving packets, it should continue listening for $N_{sniff\ timeout}$ RX slots or remaining of the $N_{sniff\ attempt}$ number of RX slots, whichever is greater.

To enter the sniff mode, the master shall issue a sniff command via the LM protocol. This message will contain the sniff interval T_{sniff} and an offset $D_{sniff.}$ The timing of the sniff mode is then determined similar as for the SCO links. In addition, an initialization flag indicates whether initialization procedure 1 or 2 is being used. The master uses initialization 1 when the MSB of the current master clock (CLK_{27}) is 0; it uses initialization 2 when the MSB of the current master clock (CLK_{27}) is 1. The slave shall apply the initialization method as indicated by the initialization flag irrespective of its clock bit value CLK_{27}. The master-to-slave sniff slots determined by the master and the slave shall be initialized on the slots for which the clock satisfies the following equation

$$CLK_{27\text{-}1} \bmod T_{sniff} = D_{sniff} \qquad \text{for initialization 1}$$

$$(\overline{CLK}_{27}, CLK_{26\text{-}1}) \bmod T_{sniff} = D_{sniff} \qquad \text{for initialization 2}$$

The slave-to-master sniff slot determined by the master and the slave shall be initialized on the slots after the master-to-slave sniff slot defined above. After initialization, the clock value CLK(k+1) for the next master-to-slave SNIFF slot is found by adding the fixed interval T_{sniff} to the clock value of the current master-to-slave sniff slot:

$$CLK(k+1) = CLK(k) + T_{sniff}$$

10.8.3 Hold mode

During the **CONNECTION** state, the ACL link to a slave can be put in a **hold** mode. This means that the slave temporarily does not support ACL packets on the channel any more (note: possible SCO links will still be supported). With the **hold** mode, capacity can be made free to do other things like scanning, paging, inquiring, or attending another piconet. The unit in **hold** mode can also enter a low-power sleep mode. During the **hold** mode, the slave unit keeps its active member address (AM_ADDR).

Prior to entering the hold mode, master and slave agree on the time duration the slave remains in the hold mode. A timer is initialized with the *holdTO* value. When the timer is expired, the slave will wake up, synchronize to the traffic on the channel and will wait for further master instructions.

10.8.4 Park mode

When a slave does not need to participate on the piconet channel, but still wants to remain synchronized to the channel, it can enter the park mode which is a low-power mode with very little activity in the slave. In the park mode, the slave gives up its active member address AM_ADDR. Instead, it receives two new addresses to be used in the park mode

- PM_ADDR: 8-bit Parked Member Address
- AR_ADDR: 8-bit Access Request Address

The PM_ADDR distinguishes a parked slave from the other parked slaves. This address is used in the master-initiated unpark procedure. In addition to the PM_ADDR, a parked slave can also be unparked by its 48-bit BD_ADDR. The all-zero PM_ADDR is a reserved address: if a parked unit has the all-zero PM_ADDR it can only be unparked by the BD_ADDR. In that case, the PM_ADDR has no meaning. The AR_ADDR is used by the slave in the slave-initiated unpark procedure. All messages sent to the parked slaves have to be carried by broadcast packets (the all-zero AM_ADDR) because of the missing AM_ADDR.

The parked slave wakes up at regular intervals to listen to the channel in order to re-synchronize and to check for broadcast messages. To support the synchronization and channel access of the parked slaves, the master supports a beacon channel described in the next section. The beacon structure is communicated to the slave when it is being parked. When the beacon structure changes, the parked slaves are updated through broadcast messages.

In addition for using it for low power consumption, the park mode is used to connect more than seven slaves to a single master. At any one time, only seven slaves can be active. However, by swapping active and parked slaves out respectively in the piconet, the number of slave virtually connected can be much larger (255 if the PM_ADDR is used, and even a larger number if the BD_ADDR is used). There is no limitation to the number of slaves that can be parked.

10.8.4.1 Beacon channel

To support parked slaves, the master establishes a beacon channel when one or more slaves are parked. The beacon channel consists of one beacon slot or a train of equidistant beacon slots which is transmitted periodically with a constant time interval. The beacon channel is illustrated in Figure 10.8 on page 117. A train of N_B ($N_B \geq 1$) beacon slots is defined with an interval of T_B slots.

The beacon slots in the train are separated by Δ_B. The start of the first beacon slot is referred to as the **beacon instant** and serves as the beacon timing reference. The beacon parameters N_B and T_B are chosen such that there are sufficient beacon slots for a parked slave to synchronize to during a certain time window in an error-prone environment.

When parked, the slave will receive the beacon parameters through an LMP command. In addition, the timing of the beacon instant is indicated through the offset D_B. Like for the SCO link (see Section 3.2 on page 45), two initialization procedures 1 or 2 are used. The master uses initialization 1 when the MSB of the current master clock (CLK_{27}) is 0; it uses initialization 2 when the MSB of the current master clock (CLK_{27}) is 1. The chosen initialization procedure is also carried by an initialization flag in the LMP command. The slave shall apply the initiations method as indicated by the initialization flag irrespective of its clock bit CLK_{27}. The master-to-slave slot positioned at the beacon instant shall be initialized on the slots for which the clock satisfies the following equation

$$CLK_{27\text{-}1} \bmod T_B = D_B \qquad \text{for initialization 1}$$

$$(\overline{CLK_{27}}, CLK_{26\text{-}1}) \bmod T_B = D_B \qquad \text{for initialization 2}$$

After initialization, the clock value CLK(k+1) for the next beacon instant is found by adding the fixed interval T_B to the clock value of the current beacon instant:

$$CLK(k+1) = CLK(k) + T_B$$

The beacon channel serves four purposes:

1. transmission of master-to-slave packets which the parked slaves can use for re-synchronization
2. carrying messages to the parked slaves to change the beacon parameters
3. carrying general broadcast messages to the parked slaves
4. unparking of one or more parked slaves

Since a slave can synchronize to any packet which is preceded by the proper channel access code, the packets carried on the beacon slots do not have to contain specific broadcast packets for parked slaves to be able to synchronize; any packet can be used. The only requirement placed on the beacon slots is that there is master-to-slave transmission present. If there is no information to be sent, **NULL** packets can be transmitted by the master. If there is indeed broadcast information to be sent to the parked slaves, the first packet of the broadcast message shall be repeated in every beacon slot of the beacon train. However, synchronous traffic like on the SCO link, may interrupt the beacon transmission.

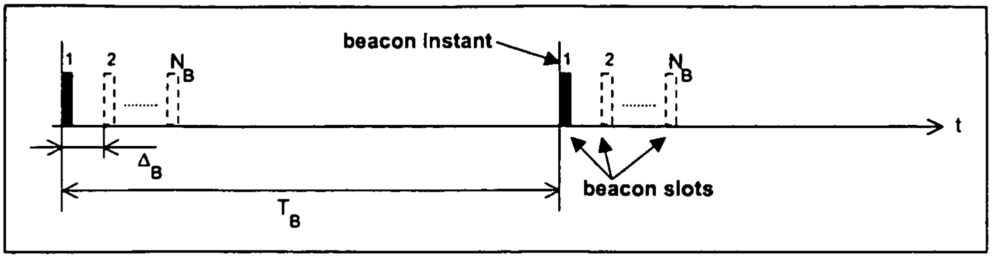


Figure 10.8: General beacon channel format

10.8.4.2 Beacon access window

In addition to the beacon slots, an access window is defined where the parked slaves can send requests to be unparked. To increase reliability, the access window can be repeated M_{access} times ($M_{access} \geq 1$), see Figure 10.9 on page 117. The access window starts a fixed delay D_{access} after the beacon instant. The width of the access window is T_{access}.

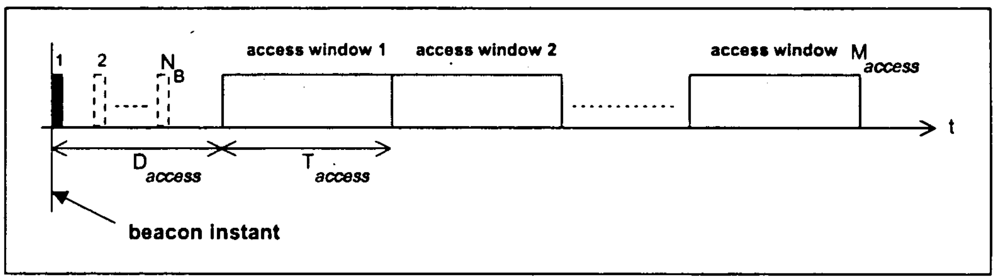


Figure 10.9: Definition of access window

The access window may support different slave access techniques, like polling, random access, or other forms of access. At this stage, only the polling technique has been defined. The format of the polling technique is shown in Figure 10.10 on page 118. The same TDD structure is used as on the piconet channel, i.e. master-to-slave transmission is alternated by slave-to-master transmission. The slave-to-master slot is divided into two half slots of 312.5 µs each. The half slot a parked slave is allowed to respond in corresponds to its access request address (AR_ADDR), see also section 10.8.4.6 on page 120. For counting the half slots to determine the access request slot, the start of the access window is used, see Figure 10.10 on page 118. The slave is only allowed to send an access request in the proper slave-to-master half slot if in the preceding master-to-slave slot a broadcast packet has been received. In this way, the master polls the parked slaves.

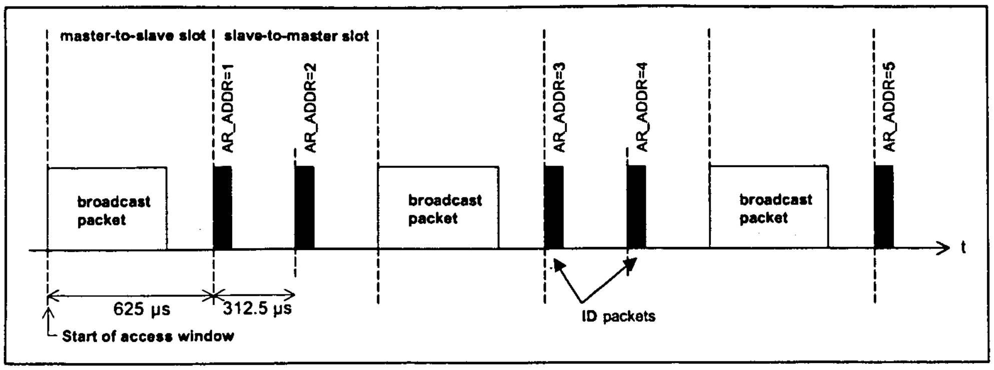


Figure 10.10: Access procedure applying the polling technique.

However, the slots of the access window can also be used for traffic on the piconet if required. For example, if an SCO connection has to be supported, the slots reserved for the SCO link may carry SCO information instead of being used for access requests, i.e. if the master-to-slave slot in the access window contains a packet different from a broadcast packet, the following slave-to-master slot cannot be used for slave access requests. Slots in the access window not affected by traffic can still be used according to the defined access structure; an example is shown in Figure 10.11 on page 118: the access procedure is continued as if no interruption had taken place.

When the slave is parked, it is indicated what type of access scheme will be used. For the polling scheme, the number of slave-to-master access slots N_{acc_slot} is indicated.

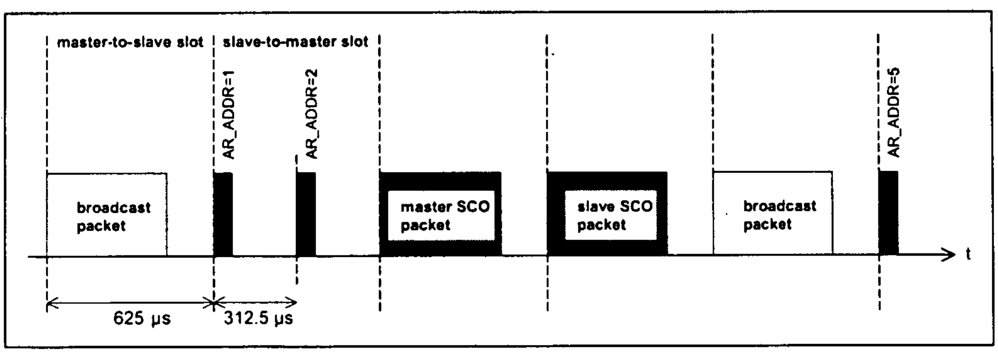


Figure 10.11: Disturbance of access window by SCO traffic

By default, the access window is always present. However, its activation depends on the master sending broadcast messages to the slave at the appropriate slots in the access window. A broadcast LMP command in the beacon slots may indicate that the access window following will not be activated. This prevents unnecessary scanning of parked slaves that want to request access.

10.8.4.3 Parked slave synchronization

Parked slaves sleep most of the time. However, periodically they wake up to re-synchronize to the channel. Any packet exchanged on the channel can be used for synchronization. Since master transmission is mandatory on the beacon slots, parked slaves will exploit the beacon channel to re-synchronize. A parked slave will wake-up at the beacon instant to read the packet sent on the first beacon slot. If this fails, it will retry on the next beacon slot in the beacon train; in total, there are N_B opportunities per beacon instant to re-synchronize. During the search, the slave may increase its search window, see also Section 9.4 on page 90. The separation between the beacon slots in the beacon train Δ_B is chosen such that consecutive search windows will not overlap.

The parked slave does not have to wake up at every beacon instant. Instead, a sleep interval can be applied which is longer than the beacon interval T_B, see Figure 10.12 on page 119. The slave sleep window must be a multiple N_{B_sleep} of T_B. The precise beacon instant the slave shall wake up on is indicated by the master with D_{B_sleep} which indicates the offset (in multiples of T_B) with respect to the beacon instant ($0< D_{B\_sleep}<N_{B\_sleep}-1$). To initialize the wake-up period, the following equations are used:

$$CLK_{27-1} \bmod (N_{B\_sleep} \bullet T_B) = D_B + D_{B\_sleep} \bullet T_B \quad \text{for initialization 1}$$

$$(\overline{CLK_{27}}, CLK_{26-1}) \bmod (N_{B\_sleep} \bullet T_B) = D_B + D_{B\_sleep} \bullet T_B \quad \text{for initialization 2}$$

where initialization 1 is chosen by the master if the MSB in the current master clock is 0 and initialization 2 is chosen if the MSB in the current master clock is 1.

When the master wants to send broadcast messages to the parked slaves, it may use the beacon slots for these broadcast messages. However, if $N_B<N_{BC}$, the slots following the last beacon slot in the beacon train shall be used for the remaining $N_{BC}-N_B$ broadcast packets. If $N_B>N_{BC}$, the broadcast message is repeated on all N_B beacon slots.

A parked slave shall at least read the broadcast messages sent in the beacon slot(s) it wakes up in; the minimum wake-up activity is to read the channel access code for re-synchronization and the packet header to check for broadcast messages.

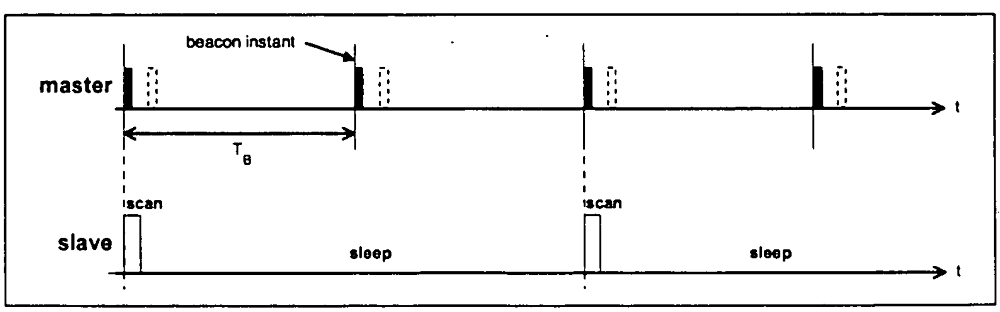


Figure 10.12: Extended sleep interval of parked slaves.

10.8.4.4 Parking

A master can park an active slave through the exchange of one or a few LMP commands. Before put into the park mode, the slave is assigned a PM_ADDR and an AR_ADDR. Every parked slave has a unique PM_ADDR; however, the AR_ADDR is not necessarily unique. Also, the beacon parameters are given by the master when the slave is parked. The slave then gives up its AM_ADDR and enters the park mode. A master can park only a single slave at a time. The park message is carried with a normal data packet and addresses the slave through its AM_ADDR.

10.8.4.5 Master-activated unparking

The master can unpark a parked slave by sending a dedicated LMP unpark command including the parked slave's address. This message is sent in a broadcast packet on the beacon slots. Either the slave's PM_ADDR is used, or its full BD_ADDR is used. The message also includes the active member address AM_ADDR the slave will use after it has re-entered the piconet. The unpark message can include a number of slave addresses so that multiple slaves can be unparked simultaneously. For each slave, a different AM_ADDR is assigned.

After having received the unpark message, the parked slave matching the PM_ADDR or BD_ADDR will leave the park mode and enter the active mode. It will keep listening to the master until it is addressed by the master through its AM_ADDR. The first packet sent by the master should be a POLL packet. The return packet in response to the POLL packet confirms that the slave has been unparked. If no response packets from the slave is received for *newconnectionTO* number of slots after the end of beacon repetition period, the master will unpark the slave again. If the slave does not receive the POLL packet for *newconnectionTO* number of slots after the end of beacon repetition period, it will return to park, with the same beacon parameters. After confirming that the slave is active, the master decides in which mode the slave will continue.

10.8.4.6 Slave-activated unparking

A slave can request access to the channel through the access window defined in section 10.8.4.2 on page 117. As shown in Figure 10.10 on page 118, the access window includes several slave-to-master half slots where the slave can send an access request message. The specific half slot the slave is allowed to respond in, corresponds to its access request address (AR_ADDR) which it has received when it was parked. The order of the half slots (in Figure 10.10 the AR_ADDR numbers linearly increase from 1 to 5) is not fixed: an LMP command sent in the beacon slots may reconfigure the access window. When a slave desires access to the channel, it sends an access request message in the proper slave-to-master half slot.The access request message of the slave is the **ID** packet containing the device access code (DAC) of the master (which is in this case the channel access code without the trailer). The parked slave is

only allowed to transmit an access request message in the half slot when in the preceding master-to-slave slot, a broadcast packet has been received. This broadcast message can contain any kind of broadcast information not necessarily related to the parked slave(s). If no broadcast information is available, a broadcast **NULL** or broadcast **POLL** packet shall be sent.

After having sent an access request, the parked slave will listen for an unpark message from the master. As long as no unpark message is received, the slave will repeat the access requests in the subsequent access windows. After the last access window (there are M_{access} windows in total, see Section 10.8.4.2 on page 117), the parked slave shall listen for an additional N_{poll} time slots for an unpark message. If no unpark message is received within N_{poll} slots after the end of the last access window, the slave may return to sleep and retry an access attempt after the next beacon instant.

After having received the unpark message, the parked slave matching the PM_ADDR or BD_ADDR will leave the park mode and enter the active mode. It will keep listening to the master until it is addressed by the master through its AM_ADDR. The first packet sent by the master should be a POLL packet. The return packet in response to the POLL packet confirms that the slave has been unparked. If no response packet from the slave is received for *newconnectionTO* number of slots after N_{poll} slots after the end of the last access window, the master will send the unpark message to the slave again. If the slave does not receive the POLL packet for *newconnectionTO* number of slots after N_{poll} slots after the end of the last access window, it will return to park, with the same beacon parameters. After confirming that the slave is active, the master decides in which mode the slave will continue.

10.8.4.7 Broadcast scan window

In the beacon train, the master can support broadcast messages to the parked slaves. However, it may extend its broadcast capacity by indicating to the parked slaves that more broadcast information is following after the beacon train. This is achieved by a special LMP command ordering the parked slaves (as well as the active slaves) to listen to the channel for broadcast messages during a limited time window. This time window starts at the beacon instant and continues for the period as indicated in the LMP command sent in the beacon train.

10.8.5 Polling schemes

10.8.5.1 Polling in active mode

The master always has full control over the piconet. Due to the stringent TDD scheme, slaves can only communicate with the master and not to other slaves. In order to avoid collisions on the ACL link, a slave is only allowed to transmit in the slave-to-master slot when addressed by the AM_ADDR in the packet

header in the preceding master-to-slave slot. If the AM_ADDR in the preceding slot does not match, or an AM_ADDR cannot be derived from the preceding slot, the slave is not allowed to transmit.

On the SCO links, the polling rule is slightly modified. The slave is allowed to transmit in the slot reserved for his SCO link unless the (valid) AM_ADDR in the preceding slot indicates a different slave. If no valid AM_ADDR can be derived in the preceding slot, the slave is still allowed to transmit in the reserved SCO slot.

10.8.5.2 Polling in park mode

In the park mode, parked slaves are allowed to send access requests in the access window provided a broadcast packet is received in the preceding master-to-slave slot. Slaves in active mode will not send in the slave-to-master slots following the broadcast packet since they are only allowed to send if addressed specifically.

10.8.6 Slot reservation scheme

The SCO link is established by negotiations between the link managers which involves the exchange of important SCO timing parameters like T_{SCO} and D_{SCO} through LMP messages.

10.8.7 Broadcast scheme

The master of the piconet can broadcast messages which will reach all slaves. A broadcast packet is characterized by the all-zero AM_ADDR. Each new broadcast message (which may be carried by a number of packets) shall start with the flush indication (L_CH=10).

A broadcast packet is never acknowledged. In an error-prone environment, the master may carry out a number of retransmissions N_{BC} to increase the probability for error-free delivery, see also Section 5.3.5 on page 72.

In order to support the **park** mode (as described in Section 10.8.4 on page 115), a master transmission shall take place at fixed intervals. This master transmission will act as a beacon to which slaves can synchronize. If no traffic takes place at the beacon event, broadcast packets shall be sent. More information is given in Section 10.8.4 on page 115.

10.9 SCATTERNET

10.9.1 General

Multiple piconets may cover the same area. Since each piconet has a different master, the piconets hop independently, each with their own channel hopping

sequence and phase as determined by the respective master. In addition, the packets carried on the channels are preceded by different channel access codes as determined by the master device addresses. As more piconets are added, the probability of collisions increases; a graceful degradation of performance results as is common in frequency-hopping spread spectrum systems.

If multiple piconets cover the same area, a unit can participate in two or more overlaying piconets by applying time multiplexing. To participate on the proper channel, it should use the associated master device address and proper clock offset to obtain the correct phase. A Bluetooth unit can act as a slave in several piconets, but only as a master in a single piconet: since two piconets with the same master are synchronized and use the same hopping sequence, they are one and the same piconet. A group of piconets in which connections consists between different piconets is called a **scatternet**.

A master or slave can become a slave in another piconet by being paged by the master of this other piconet. On the other hand, a unit participating in one piconet can page the master or slave of another piconet. Since the paging unit always starts out as master, a master-slave role exchange is required if a slave role is desired. This is described in the section 10.9.3 on page 123.

10.9.2 Inter-piconet communications

Time multiplexing must be used to switch between piconets. In case of ACL links only, a unit can request to enter the **hold** or **park** mode in the current piconet during which time it may join another piconet by just changing the channel parameters. Units in the **sniff** mode may have sufficient time to visit another piconet in between the sniff slots. If SCO links are established, other piconets can only be visited in the non-reserved slots in between. This is only possible if there is a single SCO link using **HV3** packets. In the four slots in between, one other piconet can be visited. Since the multiple piconets are not synchronized, guard time must be left to account for misalignment. This means that only 2 slots can effectively be used to visit another piconet in between the **HV3** packets.

Since the clocks of two masters of different piconets are not synchronized, a slave unit participating in two piconets has to take care of two offsets that, added to its own native clock, create one or the other master clock. Since the two master clocks drift independently, regular updates of the offsets are required in order for the slave unit to keep synchronization to both masters.

10.9.3 Master-slave switch

In principle, the unit that creates the piconet is the master. However, a master-slave (MS) switch can take place when a slave wants to become a master. For the two units involved in the switch, the MS switch results in a reversal of their TX and RX timing: a TDD switch. However, since the piconet parameters are derived from the device address and clock of the master, a master-slave switch inherently involves a redefinition of the piconet as well: a piconet switch. The

new piconet's parameters are derived from the former slave's device address and clock. As a consequence of this piconet switch, other slaves in the piconet not involved in the switch have to be moved to the new piconet, changing their timing and their hopping scheme. The new piconet parameters have to be communicated to each slave. The scenario to achieve this is described below. Assume unit A wants to become master; unit B was the former master. The following steps are taken.

- Slave A and master B agree to exchange roles.
- When confirmed by both units, both slave A and master B do the TDD switch but keep the former hopping scheme (still using the device address and clock of unit B), so there is no piconet switch yet.
- Unit A is now the master of the piconet. Since the old and new masters' clocks are asynchronous, the 1.25 ms resolution of the clock information given in the FHS packet is not sufficient for aligning the slot boundaries of the two piconets. Prior to sending the FHS packet, the new master A sends an LMP packet giving the delay between the start of the master-to-slave slots of the old and new piconet channels. This timing information ranges from 0 to 1249 μs with a resolution of 1 μs. It is used together with the clock information in the FHS packet to accurately position the correlation window when switching to the new master's timing after acknowledgment of the FHS packet.
- After the time alignment LMP message, Master A sends an FHS packet including the new AM_ADDR to slave B (the AM_ADDR in the FHS packet header is the all-zero address) still using the "old" piconet parameters. After the FHS acknowledgement, which consists of the **ID** packet and is sent by the slave on the old hopping sequence, both master A and slave B turn to the new channel parameters of the new piconet as indicated by the FHS and time alignment LMP packets (at least for the A-B connection).
- A piconet switch is enforced on each slave separately. Master A sends a time alignment and an FHS packets and waits for an acknowledgement. Transmission of the FHS packet and the acknowledgement continues on the "old" piconet parameters of unit B (compare this to the page hopping scheme used during connection establishment, see Section 10.6.4 on page 104). After FHS acknowledgement using an **ID** packet sent by the slave, the communication to this slave continues with the new device address and clock of unit A. The FHS packet sent to each slave has the old AM_ADDR in the FHS packet header and their new AM_ADDR in the FHS packet payload (the new AM_ADDR may be identical to the old AM_ADDR).
- After reception of the FHS packet acknowledgement, the new master A switches to its own timing and sends a POLL packet to verify the switch. Both the master and the slave will start a timer with a time out of *newconnectionTO* on FHS packet acknowledgement. If no response is received, the master resends the POLL packet until *newconnectionTO* is reached. After this timeout both the slave and the master return to the old piconet timing (but the TDD switch remains).The master sends the FHS packet again and the procedure is repeated.

- The new master repeats the above procedure for each slave in the old piconet.

Summarized, the MS-switch takes place in two steps: first a TDD switch of the considered master and slave, and then a piconet switch of all participants. When all slaves have acknowledged the reception of the FHS packet, each unit uses the new piconet parameters defined by the new master and the piconet switch is a fact. The information on the AM_ADDR, PM_ADDR, and other features of the old slaves is transferred from the old master to the new master. The transfer procedure is outside the scope of this procedure. Parked slaves shall be activated (using the old park parameters), be changed to the new piconet parameters, and then return to the **park** mode using the new park parameters.

10.10 POWER MANAGEMENT

Features are included into Bluetooth to ensure a low-power operation. These features are both at the microscopic level when handling the packets, and at the macroscopic level using certain operation modes.

10.10.1 Packet handling

In order to minimize power consumption, packet handling is minimized both at TX and RX sides. At the TX side, power is minimized by only sending useful data. This means that if only link control information needs to be exchanged, **NULL** packets will be used. No transmission is carried out at all if there is no link control information or involves a NAK only (NAK is implicit on no reply). If there is data to be sent, the payload length is adapted in order to send only the valid data bytes. At the RX side, packet processing takes place in different steps. If no valid access code is found in the search window, the transceiver returns to sleep. If an access code is found, the receiver unit is woken up and starts to process the packet header. If the HEC fails, the unit will return to sleep after the packet header. A valid header will indicate if a payload will follow and how many slots are involved.

10.10.2 Slot occupancy

As was described in Section 4.4 on page 54, the packet type indicates how many slots a packet may occupy. A slave not addressed in the first slot can go to sleep for the remaining slots the packet may occupy. This can be read from the TYPE code.

10.10.3 Low-power modes

In Section 10.8 on page 112, three modes were described during the **CONNECTION** state which reduce power consumption. If we list the modes in increasing order of power efficiency then the **sniff** mode has the higher duty

cycle, followed by the **h ld** mode with a lower duty cycle, and finishing with the **park** mode with the lowest duty cycle.

10.11 LINK SUPERVISION

A connection may break down due to various reasons such as a device moving out of range or a power failure condition. Since this may happen without any prior warning, it is important to monitor the link on both the master and the slave side to avoid possible collisions when the AM_ADDR is reassigned to another slave.

To be able to supervise link loss, both the master and the slave use link supervision timers, $T_{supervision}$. Upon reception of a packet that passes the HEC check and has the correct AM_ADDR, the timer is reset. If at any time in connection state, the timer reaches the *supervisionTO* value, the connection is reset. The same timeout value is used for both SCO and ACL connections.

The timeout period, *supervisionTO,* is negotiated at the LM level. Its value is chosen so that the supervision timeout will be longer than hold and sniff periods. Link supervision of a parked slave will be done by unparking and re-parking the slave.

11 HOP SELECTION

In total, 10 types of hopping sequences are defined – five for the 79-hop and five for the 23-hop system, respectively. Using the notation of parentheses () for figures related to the 23-hop system, these sequences are:

- A **page hopping sequence** with 32 (16) unique wake-up frequencies distributed equally over the 79 (23) MHz, with a period length of 32 (16);
- A **page response sequence** covering 32 (16) unique response frequencies that all are in an one-to-one correspondence to the current page hopping sequence. The master and slave use different rules to obtain the same sequence;
- An **inquiry sequence** with 32 (16) unique wake-up frequencies distributed equally over the 79 (23) MHz, with a period length of 32 (16);
- A **inquiry response sequence** covering 32 (16) unique response frequencies that all are in an one-to-one correspondence to the current inquiry hopping sequence.
- A **channel hopping sequence** which has a very long period length, which does not show repetitive patterns over a short time interval, but which distributes the hop frequencies equally over the 79 (23) MHz during a short time interval;

For the page hopping sequence, it is important that we can easily shift the phase forward or backward, so we need a 1-1 mapping from a counter to the hop frequencies. For each case, both a hop sequence from master to slave and from slave to master are required.

The inquiry and inquiry response sequences always utilizes the GIAC LAP as lower address part and the DCI (Section 5.4 on page 73) as upper address part in deriving the hopping sequence, even if it concerns a DIAC inquiry.

11.1 GENERAL SELECTION SCHEME

The selection scheme consists of two parts:

- selecting a sequence;
- mapping this sequence on the hop frequencies;

The general block diagram of the hop selection scheme is shown in Figure 11.1 on page 128. The mapping from the input to a particular hop frequency is performed in the selection box. Basically, the input is the native clock and the current address. In **CONNECTION** state, the native clock (CLKN) is modified by an offset to equal the master clock (CLK). Only the 27 MSBs of the clock are used. In the **page** and **inquiry** substates, all 28 bits of the clock are used. However, in **page** substate the native clock will be modified to the master's estimate of the paged unit.

The address input consists of 28 bits, i.e., the entire LAP and the 4 LSBs of the UAP. In **CONNECTION** state, the address of the master is used. In **pag** sub-state the address of the paged unit is used. When in **inquiry** substate, the UAP/LAP corresponding to the GIAC is used. The output constitutes a pseudo-random sequence, either covering 79 hop or 23 hops, depending on the state.

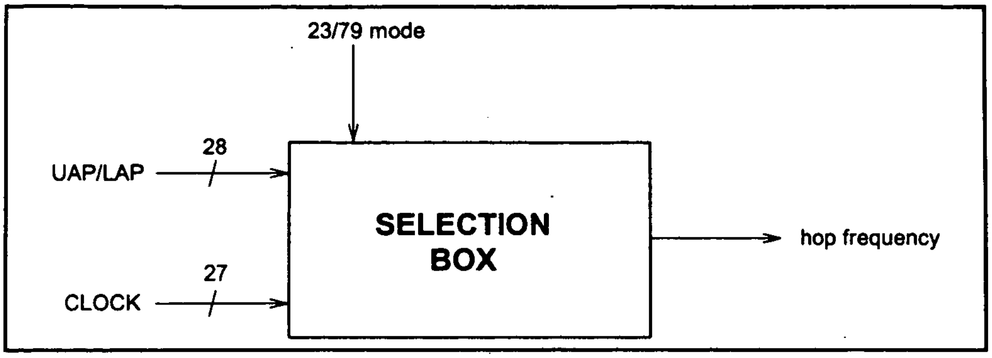


Figure 11.1: General block diagram of hop selection scheme.

For the 79-hop system, the selection scheme chooses a segment of 32 hop frequencies spanning about 64 MHz and visits these hops once in a random order. Next, a different 32-hop segment is chosen, etc. In case of the **page, page scan,** or **page response** substates, the same 32-hop segment is used all the time (the segment is selected by the address; different units will have different paging segments). In connection state, the output constitutes a pseudo-random sequence that slides through the 79 hops or 23 hops, depending on the selected hop system. For the 23-hop systems, the segment size is 16. The principle is depicted in Figure 11.2

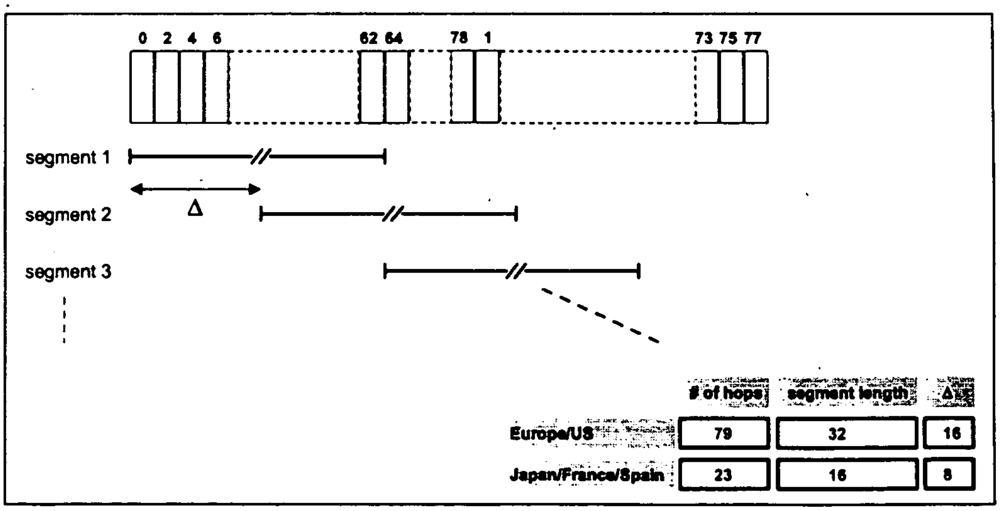


Figure 11.2: Hop selection scheme in CONNECTION state.

11.2 SELECTION KERNEL

The hop selection kernels for the 79 hop system and the 23 hop system are shown in Figure 11.3 on page 129 and Figure 11.4 on page 129, respectively. The X input determines the phase in the 32-hop segment, whereas Y1 and Y2 selects between master-to-slave and slave-to-master transmission. The inputs A to D determine the ordering within the segment, the inputs E and F determine the mapping onto the hop frequencies. The kernel addresses a register containing the hop frequencies. This list should be created such that first all even hop frequencies are listed and then all odd hop frequencies. In this way, a 32-hop segment spans about 64 MHz, whereas a 16-hop segment spans the entire 23-MHz.

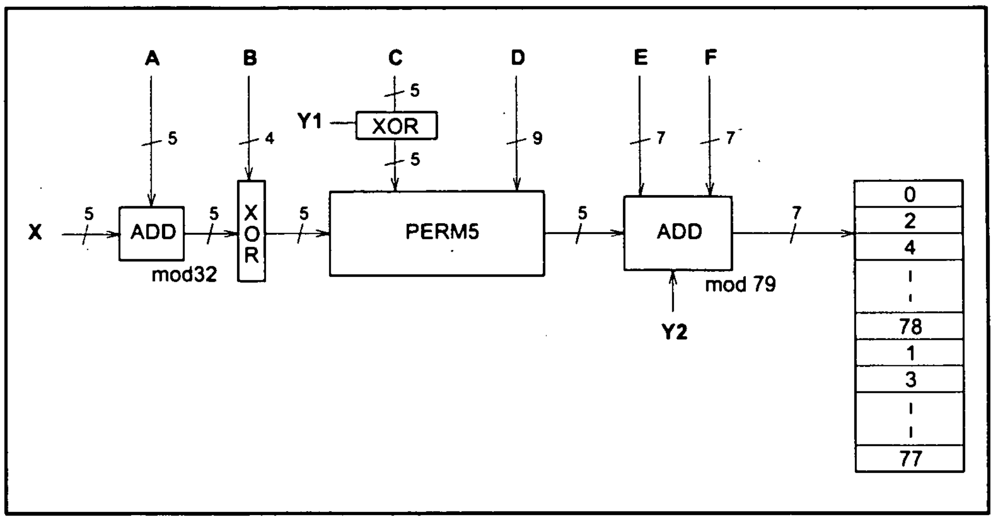


Figure 11.3: Block diagram of hop selection kernel for the 79-hop system.

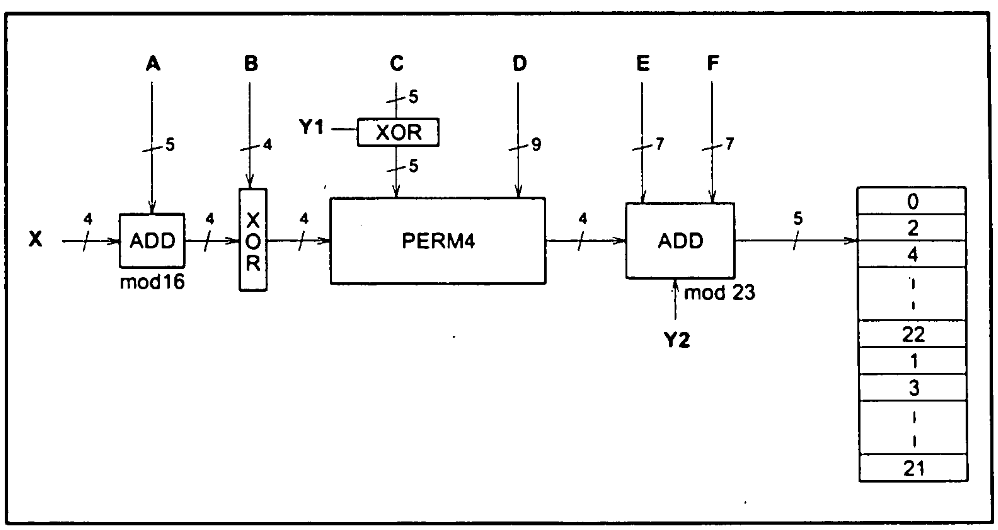


Figure 11.4: Block diagram of hop selection kernel for the 23-hop system.

The selection procedure consists of an addition, an XOR operation, a permutation operation, an addition, and finally a register selection. In the remainder of this chapter, the notation A_i is used for bit i of the BD_ADDR.

11.2.1 First addition operation

The first addition operation only adds a constant to the phase and applies a modulo 32 or a modulo 16 operation. For the page hopping sequence, the first addition is redundant since it only changes the phase within the segment. However, when different segments are concatenated (as in the channel hopping sequence), the first addition operation will have an impact on the resulting sequence.

11.2.2 XOR operation

Let Z' denote the output of the first addition. In the XOR operation, the four LSBs of Z' are modulo-2 added to the address bits $A_{22\text{-}19}$. The operation is illustrated in Figure 11.5 on page 130.

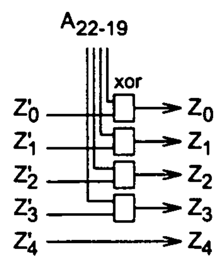


Figure 11.5: XOR operation for the 79-hop system. The 23-hop system is the same except for the Z'_4/Z_4 wire that does not exist.

11.2.3 Permutation operation

The permutation operation involves the switching from 5 inputs to 5 outputs for the 79 hop system and from 4 inputs to 4 outputs for 23 hop system, in a manner controlled by the control word. In Figure 11.6 on page 132 and Figure 11.7 on page 132 the permutation or switching box is shown. It consists of 7 stages of butterfly operations. Table 11.1 and Table 11.2 shows the control of the butterflies by the control signals P. Note that $P_{0\text{-}8}$ corresponds to $D_{0\text{-}8}$, and, P_{i+9} corresponds to $C_i \oplus Y1$ for $i = 0 \ldots 4$ in Figure 11.3 and Figure 11.4.

Control signal	Butterfly	Control signal	Butterfly
P_0	$\{Z_0,Z_1\}$	P_8	{Z1,Z4}
P_1	$\{Z_2,Z_3\}$	P_9	$\{Z_0,Z_3\}$
P_2	$\{Z_1,Z_2\}$	P_{10}	$\{Z_2,Z_4\}$
P_3	$\{Z_3,Z_4\}$	P_{11}	$\{Z_1,Z_3\}$
P_4	$\{Z_0,Z_4\}$	P_{12}	$\{Z_0,Z_3\}$
P_5	$\{Z_1,Z_3\}$	P_{13}	$\{Z_1,Z_2\}$
P_6	$\{Z_0,Z_2\}$		
P_7	$\{Z_3,Z_4\}$		

Table 11.1: Control of the butterflies for the 79 hop system

Control signal	Butterfly	Control signal	Butterfly
P_0	$\{Z_0,Z_1\}$	P_8	$\{Z_0,Z_2\}$
P_1	$\{Z_2,Z_3\}$	P_9	$\{Z_1,Z_3\}$
P_2	$\{Z_0,Z_3\}$	P_{10}	$\{Z_0,Z_3\}$
P_3	$\{Z_1,Z_2\}$	P_{11}	$\{Z_1,Z_2\}$
P_4	$\{Z_0,Z_2\}$	P_{12}	$\{Z_0,Z_1\}$
P_5	$\{Z_1,Z_3\}$	P_{13}	$\{Z_2,Z_3\}$
P_6	$\{Z_0,Z_1\}$		
P_7	$\{Z_2,Z_3\}$		

Table 11.2: Control of the butterflies for the 23 hop system

The Z input is the output of the XOR operation as described in the previous section. The butterfly operation can be implemented with multiplexers as depicted in Figure 11.8 on page 132.

Figure 11.7: Permutation operation for the 23 hop system.

Figure 11.8: Butterfly implementation.

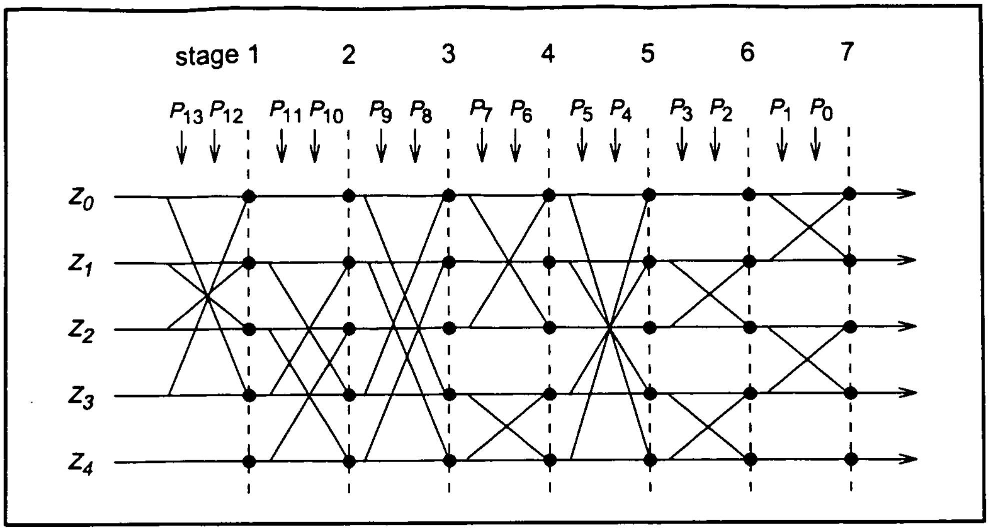
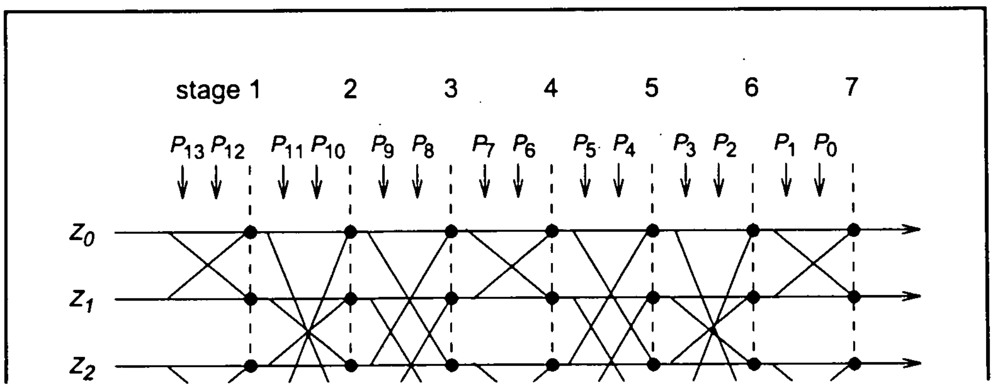


Figure 11.6: Permutation operation for the 79 hop system.



11.2.4 Second addition operation

The addition operation only adds a constant to the output of the permutation operation. As a result, the 16-hop or 32-hop segment is mapped differently on the hop frequencies. The addition is applied modulo 79 or modulo 23 depending on the system type (Europe/US vs. others).

11.2.5 Register bank

The output of the adder addresses a bank of 79 or 23 registers. The registers are loaded with the synthesizer code words corresponding to the hop frequencies 0 to 78 or 0 to 22. Note that the upper half of the bank contains the even hop frequencies, whereas the lower half of the bank contains the odd hop frequencies.

11.3 CONTROL WORD

In the following section X_{j-i}, $i<j$, will denote bits i, $i+1$,...,j of the bit vector X. By convention, X_0 is the least significant bit of the vector X.

The control word P of the kernel is controlled by the overall control signals X, Y1, Y2, and A to F as illustrated in Figure 11.3 on page 129 and Figure 11.4 on page 129. During paging and inquiry, the inputs A to E use the address values as given in the corresponding columns of Table 11.3 on page 134 and Table 11.4 on page 134. In addition, the inputs X, Y1 and Y2 are used. The F input is unused. In the 79-hop system, the clock bits CLK_{6-2} (i.e., input X) specifies the phase within the length 32 sequence, while for the 23-hop system, CLK_{5-2} specifies the phase within the length 16 sequence. For both systems, CLK_1 (i.e., inputs Y1 and Y2) is used to select between TX and RX. The address inputs determine the sequence order within segments. The final mapping onto the hop frequencies is determined by the register contents.

In the following we will distinguish between three types of clocks: the piconet's master clock, the Bluetooth unit's native clock, and the clock estimate of a paged Bluetooth unit. These types are marked in the following way:

1. CLK_{27-0}: Master clock of the current piconet.
2. $CLKN_{27-0}$: Native clock of the unit.
3. $CLKE_{27-0}$: The paging unit's estimate of the paged unit's native clock.

During the **CONNECTION** state, the inputs A, C and D result from the address bits being bit-wise XORed with the clock bits as shown in the "Connection state" column of Table 11.3 on page 134 and Table 11.4 on page 134 (the two MSBs are XORed together, the two second MSBs are XORed together, etc.). Consequently, after every 32 (16) time slots, a new length 32 (16) segment is selected in the 79-hop (23-hop) system. The sequence order within a specific

segment will not be repeated for a very long period. Thus, the overall hopping sequence consists of concatenated segments of 32-hops each. Since each 32-hop sequence spans more than 80% of the 79 MHz band, the desired frequency spreading over a short time interval is obtained.

	Page scan/ Inquiry scan	Page/Inquiry	Page response (master/slave) and Inquiry response	Connection state
X	$CLKN_{16-12}$	$Xp^{(79)}_{4-0}/Xi^{(79)}_{4-0}$	$Xprm^{(79)}_{4-0}/Xprs^{(79)}_{4-0}$	CLK_{6-2}
Y1	0	$CLKE_1/CLKN_1$	$CLKE_1/CLKN_1$	CLK_1
Y2	0	$32 \times CLKE_1/$ $32 \times CLKN_1$	$32 \times CLKE_1/$ $32 \times CLKN_1$	$32 \times CLK_1$
A	A_{27-23}	A_{27-23}	A_{27-23}	$A_{27-23} \oplus CLK_{25-21}$
B	A_{22-19}	A_{22-19}	A_{22-19}	A_{22-19}
C	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0} \oplus CLK_{20-16}$
D	A_{18-10}	A_{18-10}	A_{18-10}	$A_{18-10} \oplus CLK_{15-7}$
E	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$
F	0	0	0	$16 \times CLK_{27-7} \bmod 79$

Table 11.3: Control for 79-hop system.

	Page scan/ Inquiry scan	Page/Inquiry	Page response (master/slave) and Inquiry response	Connection state
X	$CLKN_{15-12}$	$Xp^{(23)}_{3-0}/Xi^{(23)}_{3-0}$	$Xprm^{(23)}_{3-0}/Xprs^{(23)}_{3-0}$	CLK_{5-2}
Y1	0	$CLKE_1/CLKN_1$	$CLKE_1/CLKN_1$	CLK_1
Y2	0	$16 \times CLKE_1/$ $16 \times CLKN_1$	$16 \times CLKE_1/$ $16 \times CLKN_1$	$16 \times CLK_1$
A	A_{27-23}	A_{27-23}	A_{27-23}	$A_{27-23} \oplus CLK_{25-21}$
B	A_{22-19}	A_{22-19}	A_{22-19}	A_{22-19}
C	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0} \oplus CLK_{20-16}$
D	A_{18-10}	A_{18-10}	A_{18-10}	$A_{18-10} \oplus CLK_{15-7}$
E	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$
F	0	0	0	$8 \times CLK_{27-6} \bmod 23$

Table 11.4: Control for 23-hop system.

11.3.1 Page scan and Inquiry scan substates

In **page scan**, the Bluetooth device address of the scanning unit is used as address input. In **inquiry scan**, the GIAC LAP and the four LSBs of the DCI (as A_{27-24}), are used as address input for the hopping sequence. Naturally, for the transmitted access code and in the receiver correlator, the appropriate GIAC or DIAC is used. The application decides which inquiry access code to use depending on the purpose of the inquiry.

The five X input bits vary depending on the current state of the unit. In the **page scan** and **inquiry scan** substates, the native clock (CLKN) is used. In **CONNECTION** state the master clock (CLK) is used as input. The situation is somewhat more complicated for the other states.

11.3.2 Page substate

In the **page** substate of the 79-hop system, the paging unit shall start using the **A**-train, i.e., $\{f(k-8), \ldots, f(k), \ldots, f(k+7)\}$, where $f(k)$ is the source's estimate of the current receiver frequency in the paged unit. Clearly, the index k is a function of all the inputs in Figure 11.3. There are 32 possible paging frequencies within each 1.28 second interval. Half of these frequencies belongs to the **A**-train, the rest (i.e., $\{f(k+8), \ldots, f(k+15), f(k-16), \ldots, f(k-9)\}$) belongs to the **B**-train. In order to achieve the -8 offset of the **A**-train, a constant of 24 can be added to the clock bits (which is equivalent to -8 due to the modulo 32 operation). Clearly, the **B**-train may be accomplished by setting the offset to 8. A cyclic shift of the order within the trains is also necessary in order to avoid a possible repetitive mismatch between the paging and scanning units. Thus,

$$Xp^{(79)} = [\mathrm{CLKE}_{16-12} + k_{offset} + (\mathrm{CLKE}_{4-2,0} - \mathrm{CLKE}_{16-12}) \bmod 16] \bmod 32, \qquad \text{(EQ 2)}$$

where

$$k_{offset} = \begin{cases} 24 & \text{A-train,} \\ 8 & \text{B-train.} \end{cases} \qquad \text{(EQ 3)}$$

Alternatively, each switch between the **A**- and **B**-trains may be accomplished by adding 16 to the current value of k_{offset} (originally initialized with 24).

In the **page** substate of the 23-hop system, the paging unit makes use of the **A**-train only. A constant offset of 8 is used in order to start with $f(k-8)$. Moreover, only four bits are needed since the additions are modulo 16. Consequently,

$$Xp^{(23)} = [\mathrm{CLKE}_{15-12} + 8 + \mathrm{CLKE}_{4-2,0}] \bmod 16, \qquad \text{(EQ 4)}$$

11.3.3 Page response

11.3.3.1 Slave response

A unit in the **page scan** substate recognizing its own access code enters the **slave response** substate. In order to eliminate the possibility of loosing the link due to discrepancies of the native clock CLKN and the master's clock estimate CLKE, the four bits CLKN_{16-12} must be frozen at their current value. The value is frozen to the content it has in the slot where the recipient's access code is detected. Note that the actual native clock is *not* stopped; it is merely the values of the bits used for creating the X-input that are kept fixed for a while. In the sequel, a frozen value is marked by an asterisk (*).

For each response slot the paged unit will use an X-input value one larger (modulo 32 or 16) than in the preceding response slot. However, the first response is made with the X-input kept at the same value as it was when the access code was recognized. Let N be a counter starting at zero. Then, the X-input in the $(N+1)$-th response slot (the first response slot being the one immediately following the page slot now responding to) of the **slave response** substate becomes

$$Xprs^{(79)} = [\mathrm{CLKN}^*_{16-12} + N] \bmod 32, \qquad \text{(EQ 5)}$$

and

$$Xprs^{(23)} = [\mathrm{CLKN}^*_{15-12} + N] \bmod 16, \qquad \text{(EQ 6)}$$

for the 79-hop and 23-hop systems, respectively. The counter N is set to zero in the slot where the slave acknowledges the page (see Figure 10.6 on page 105 and Figure 10.7 on page 105). Then, the value of N is increased by one each time CLKN_1 is set to zero, which corresponds to the start of a master TX slot. The X-input is constructed this way until the first accepted **FHS** packet is received *and* the immediately following response packet has been transmitted. After this the slave enters the **CONNECTION** state using the parameters received in the **FHS** packet.

11.3.3.2 Master response

The paging unit enters **master response** substate upon receiving a slave response. Clearly, also the master must freeze its estimated slave clock to the value that triggered a response from the paged unit. It is equivalent to using the values of the clock estimate when receiving the slave response (since only CLKE_1 will differ from the corresponding page transmission). Thus, the values are frozen when the slave **ID** packet is received. In addition to the used clock bits, also the current value of k_{offset} must be frozen. The master will adjust its X-input in the same way the paged unit does, i.e., by incrementing this value by

one for each time CLKE_1 is set to zero. The first increment shall be done before sending the **FHS** packet to the paged unit. Let N be a counter starting at one. The rules for forming the X-inputs become

$$Xprm^{(79)} = [\mathrm{CLKE}^*_{16-12} + k_{offset}^* + (\mathrm{CLKE}^*_{4-2,0} - \mathrm{CLKE}^*_{16-12}) \bmod 16 + N] \bmod 32, \quad \text{(EQ 7)}$$

and

$$Xprm^{(23)} = [\mathrm{CLKE}^*_{15-12} + 8 + \mathrm{CLKE}^*_{4-2,0} + N] \bmod 16, \quad \text{(EQ 8)}$$

for the 79-hop and 23-hop systems, respectively. The value of N is increased each time CLKE_1 is set to zero, which corresponds to the start of a master TX slot.

11.3.4 Inquiry substate

The X-input of the **inquiry** substate is quite similar to what is used in the **page** substate. Since no particular unit is addressed, the native clock CLKN of the inquirer is used. Moreover, which of the two train offsets to start with is of no real concern in this state. Consequently,

$$Xi^{(79)} = [\mathrm{CLKN}_{16-12} + k_{offset} + (\mathrm{CLKN}_{4-2,0} - \mathrm{CLKN}_{16-12}) \bmod 16] \bmod 32, \quad \text{(EQ 9)}$$

where k_{offset} is defined by (EQ 3). The initial choice of the offset is arbitrary. For the 23-hop system,

$$Xi^{(23)} = [\mathrm{CLKN}_{15-12} + 8 + \mathrm{CLKN}_{4-2,0}] \bmod 16, \quad \text{(EQ 10)}$$

The GIAC LAP and the four LSBs of the DCI (as A_{27-24}) are used as address input for the hopping sequence generator.

11.3.5 Inquiry response

The **inquiry response** substate is similar to the **slave response** with respect to the X-input. Thus, (EQ 5) and (EQ 6) holds. However, the counter N is increased not on CLKN_1 basis, but rather after each **FHS** packet has been transmitted in response to the inquiry.

The GIAC LAP and the four LSBs of the DCI (as A_{27-24}) are used as address input for the hopping sequence generator. The other input bits to the generator are the same as in the case of page response.

11.3.6 Connection stat

In **CONNECTION** state, the clock bits to use in the channel hopping sequence generation are always according to the master clock, CLK. The address bits are taken from the Bluetooth device address of the master.

12 BLUETOOTH AUDIO

On the Bluetooth air-interface, either a 64 kb/s log PCM format (A-law or µ-law) is used, or a 64 kb/s CVSD (Continuous Variable Slope Delta Modulation) is used. The latter format applies an adaptive delta modulation algorithm with syllabic companding.

The voice coding on the line interface should have a quality equal to or better than the quality of 64 kb/s log PCM.

Table 12.1 on page 139 summarizes the voice coding schemes supported on the air interface. The appropriate voice coding scheme is selected after negotiations between the Link Managers.

Voice Codecs	
linear	CVSD
8-bit logarithmic	**A-law**
	µ-law

Table 12.1: Voice coding schemes supported on the air interface.

12.1 LOG PCM CODEC

Since the voice channels on the air-interface can support a 64 kb/s information stream, a 64 kb/s log PCM traffic can be used for transmission. Either A-law or µ-law compression can be applied. In the event that the line interface uses A-law and the air interface uses µ-law or vice versa, a conversion from A-law to µ-law is performed. The compression method follows ITU-T recommendations G. 711.

12.2 CVSD CODEC

A more robust format for voice over the air interface is a delta modulation. This modulation scheme follows the waveform where the output bits indicate whether the prediction value is smaller or larger then the input waveform. To reduce slope overload effects, syllabic companding is applied: the step size is adapted according to the average signal slope. The input to the CVSD encoder is 64 ksamples/s linear PCM. Block diagrams of the CVSD encoder and CVSD decoder are shown in Figure 12.1 on page 140, Figure 12.2 on page 140 and Figure 12.3 on page 140. The system is clocked at 64 kHz.

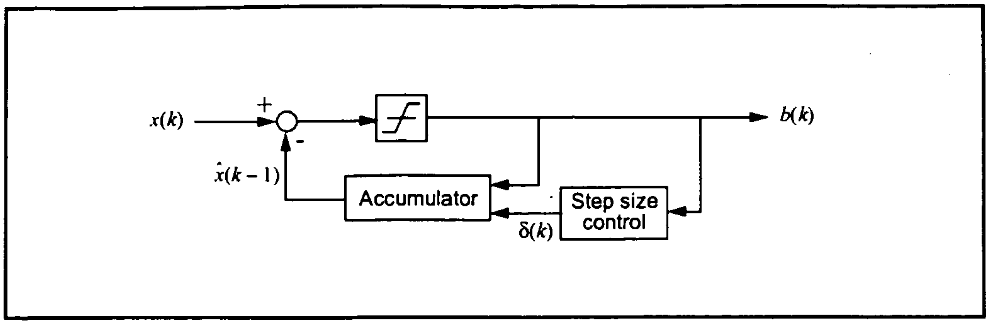


Figure 12.1: Block diagram of CVSD encoder with syllabic companding.

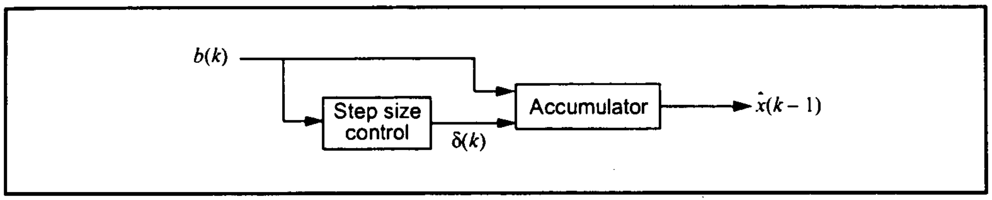


Figure 12.2: Block diagram of CVSD decoder with syllabic companding.

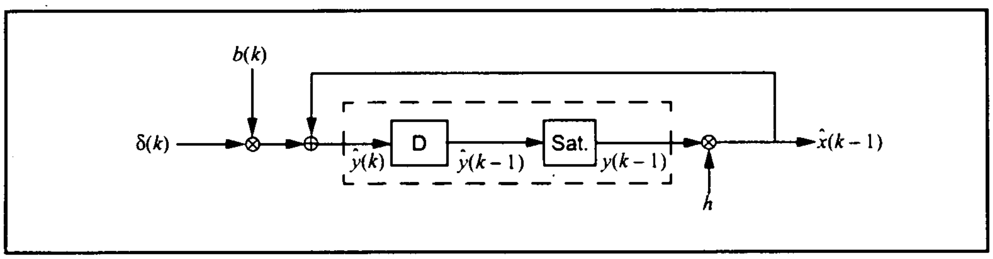


Figure 12.3: Accumulator procedure

Let $\mathrm{sgn}(x) = 1$ for $x \geq 0$, otherwise $\mathrm{sgn}(x) = -1$. On air these numbers are represented by the sign bit; i.e. negative numbers are mapped on "1" and positive numbers are mapped on "0". Denote the CVSD encoder output bit $b(k)$, the accumulator contents $y(k)$, and the step size $\delta(k)$. Furthermore, let *h* be the decay factor for the accumulator, let β denote the decay factor for the step size, and, let α be the syllabic companding parameter. The latter parameter monitors the slope by considering the *K* most recent output bits

Let

$$\hat{x}(k) = hy(k). \qquad \text{(EQ 11)}$$

Then, the CVSD encoder internal state is updated according to the following set of equations:

$$b(k) = \mathrm{sgn}\{x(k) - \hat{x}(k-1)\}, \qquad \text{(EQ 12)}$$

$$\alpha = \begin{cases} 1, & \text{if } J \text{ bits in the last } K \text{ output bits are equal,} \\ 0, & \text{otherwise,} \end{cases} \qquad \text{(EQ 13)}$$

$$\delta(k) = \begin{cases} \min\{\delta(k-1) + \delta_{min}, \delta_{max}\}, & \alpha = 1, \\ \max\{\beta\delta(k-1), \delta_{min}\}, & \alpha = 0, \end{cases} \qquad \text{(EQ 14)}$$

$$y(k) = \begin{cases} \min\{\hat{y}(k), y_{max}\}, & \hat{y}(k) \geq 0. \\ \max\{\hat{y}(k), y_{min}\}, & \hat{y}(k) < 0. \end{cases} \qquad \text{(EQ 15)}$$

where

$$\hat{y}(k) = \hat{x}(k-1) + b(k)\delta(k). \qquad \text{(EQ 16)}$$

In these equations, δ_{min} and δ_{max} are the minimum and maximum step sizes, and, y_{min} and y_{max} are the accumulator's negative and positive saturation values, respectively.

For a 64 kb/s CVSD, the parameters as shown in Table 12.2 must be used. The numbers are based on a 16 bit signed number output from the accumulator. These values result in a time constant of 0.5 ms for the accumulator decay, and a time constant of 16 ms for the step size decay

Parameter	Value
h	$1-\frac{1}{32}$
β	$1-\frac{1}{1024}$
J	4
K	4
δ_{min}	10
δ_{max}	1280
y_{min}	-2^{15} or $-2^{15}+1$
y_{max}	$2^{15}-1$

Table 12.2: CVSD parameter values. The values are based on a 16 bit signed number output from the accumulator.

12.3 ERROR HANDLING

In the **DV** and **HV3** packet, the voice is not protected by FEC. The quality of the voice in an error-prone environment then depends on the robustness of the voice coding scheme. CVSD, in particular, is rather insensitive to random bit errors, which are experienced as white background noise. However, when a packet is rejected because either the channel access code or the HEC test was unsuccessful, measures have to be taken to "fill" in the lost speech segment.

The voice payload in the **HV2** packet is protected by a 2/3 rate FEC. If errors occur which cannot be corrected, these should be ignored. That is, from the 15-bit FEC segment with uncorrected errors, the 10-bit information part as found before the FEC decoder should be used. The **HV1** packet is protected by a 3-repeat FEC. In the majority detection scheme uncorrected errors cannot occur.

12.4 GENERAL AUDIO REQUIREMENTS

These specifications are tentative and will be fixed within 18 months after the release of the Bluetooth Specification version 1.0 Draft Foundation.

12.4.1 Signal levels

For A-law and μ-law log-PCM encoded signals the requirements on signal levels follows ITU-T G.711.

Full swing at the 16 bit linear PCM interface to the CVSD encoder is defined to be 3 dBm0. A digital CVSD encoded test signal is provided in a Test Signal file available on the website. This signal is generated by a software implementation of a reference CVSD encoder. The digital encoder input signal (1020 Hz, sine-wave) generating the test signal has a nominal power of -15 dBm0. When the CVSD encoded test signal is fed through the CVSD receiver chain, the nominal output power should be -15 ± 1.0 dBm0.

12.4.2 CVSD audio quality

For Bluetooth audio quality the requirements are put on the transmitter side. The 64 ksamples/s linear PCM input signal must have negligible spectral power density above 4 kHz. A set of reference input signals are encoded by the transmitter and sent through a reference decoder (available on the website). The power spectral density in the 4-32 kHz band of the decoded signal at the 64 ksample/s linear PCM output, should be more than 20 dB below the maximum in the 0-4 kHz range.

13 BLUETOOTH ADDRESSING

13.1 BLUETOOTH DEVICE ADDRESS (BD_ADDR)

Each Bluetooth transceiver is allocated a unique 48-bit Bluetooth device address (BD_ADDR). This address is derived from the IEEE802 standard. This 48-bit address is divided into three fields:

- LAP field: lower address part consisting of 24 bits
- UAP field: upper address part consisting of 8 bits
- NAP field: non-significant address part consisting of 16 bits

The LAP and UAP form the significant part of the BD_ADDR. The total address space obtained is 2^{32}.

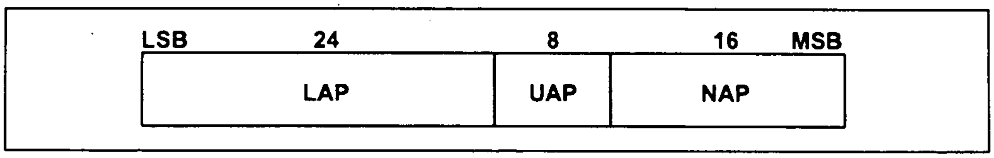


Figure 13.1: Format of BD_ADDR

13.2 ACCESS CODES

In the Bluetooth system, 72-bit and 68-bit access codes are used for signalling purposes. Three different access codes are defined, see also Section 4.2.1 on page 48:

- device access code (DAC)
- channel access code (CAC)
- inquiry access code (IAC)

There is one general IAC (GIAC) for general inquiry operations and there are 63 dedicated IACs (DIACs) for dedicated inquiry operations. All codes are derived from a LAP of the BD_ADDR. The device access code is used during page, page scan and page response substates. It is a code derived from the unit's BD_ADDR. The channel access code characterizes the channel of the piconet and forms the preamble of all packets exchanged on the channel. The channel access code is derived from the LAP of the master BD_ADDR. Finally, the inquiry access code is used in inquiry operations. A general inquiry access code is common to all Bluetooth units; a set of dedicated inquiry access codes is used to inquire for classes of devices.

The access code is also used to indicate to the receiver the arrival of a packet. It is used for timing synchronization and offset compensation. The receiver correlates against the entire sync word in the access code, providing a very robust signalling. During channel setup, the code itself is used as an ID packet to sup-

port the acquisition process. In addition, it is used during random access procedures in the PARK state.

The access code consists of preamble, sync word and a trailer, see Section 4.2 on page 48. The next two sections describe the generation of the sync word.

13.2.1 Synchronization word definition

The sync words are based on a (64,30) expurgated block code with an overlay (bit-wise XOR) of an 64 bit full length PN-sequence. The expurgated code guarantees large Hamming distance ($d_{min} = 14$) between sync words based on different addresses. The PN sequence improves the autocorrelation properties of the access code. The following steps describe how to generate the sync word:

1. Generate information sequence;
2. XOR this with the “information covering” part of the PN overlay sequence;
3. Generate the codeword;
4. XOR the codeword with all 64 bits of the PN overlay sequence;

The information sequence is generated by appending 6 bits to the 24 bit LAP (step 1). The appended bits are 001101 if the MSB of the LAP equals 0. If the MSB of the LAP is 1 the appended bits are 110010. The LAP MSB together with the appended bits constitute a length-seven Barker sequence. The purpose of including a Barker sequence is to further improve the autocorrelation properties. In step 2 the information is pre-scrambled by XORing it with the bits $p_{34}\ldots p_{63}$ of the *pseudo-random noise* (PN) sequence (defined in section 13.2.2 on page 146). After generating the codeword (step 3), the complete PN sequence is XORed to the codeword (step 4). This step de-scrambles the information part of the codeword. At the same time the parity bits of the codeword are scrambled. Consequently, the original LAP and Barker sequence are ensured a role as a part of the access code sync word, and the cyclic properties of the underlying code is removed. The principle is depicted in Figure 13.2 on page 145

In the sequel, binary sequences will be denoted by their corresponding D-transform (in which D^i represents a delay of i time units). Let $p'(D) = p'_0 + p'_1 D + \ldots + p'_{62} D^{62}$ be the 63 bit pseudo-random sequence, where p'_0 is the first bit (LSB) leaving the PRNG (see Figure 13.3 on page 147), and, p'_{62} is the last bit (MSB). To obtain 64 bits, an extra zero is appended at the *end* of this sequence (thus, $p'(D)$ is unchanged). For notational convenience, the reciprocal of this extended polynomial, $p(D) = D^{63} p'(1/D)$, will be used in the sequel. This is the sequence $p'(D)$ in reverse order. We denote the 24 bit lower

address part (LAP) of the Bluetooth address by $a(D) = a_0 + a_1 D + \ldots + a_{23} D^{23}$ (a_0 is the LSB of the Bluetooth address).

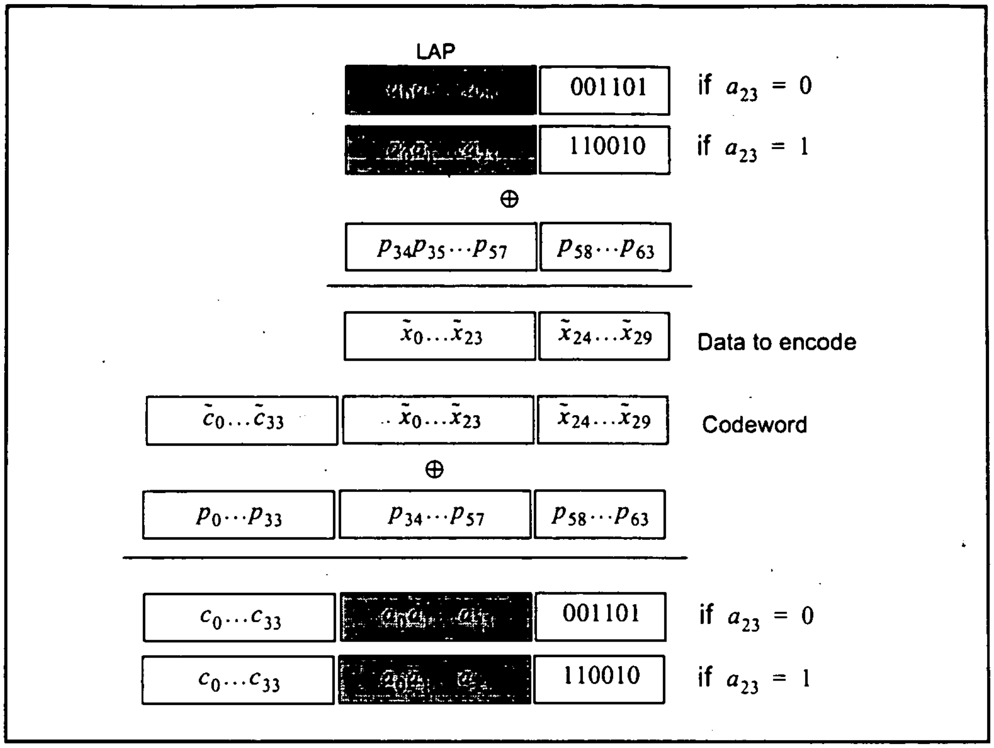


Figure 13.2: Construction of the sync word.

The (64,30) block code generator polynomial is denoted $g(D) = (1 + D)g'(D)$, where $g'(D)$ is the generator polynomial `157464165547` (octal notation) of a primitive binary (63,30) BCH code. Thus, in octal notation we have

$$g(D) = 260534236651, \qquad \text{(EQ 17)}$$

the left-most bit corresponds to the high-order (g_{34}) coefficient.The DC-free four bit sequences `0101` and `1010` can be written

$$\begin{cases} F_0(D) = D + D^3, \\ F_1(D) = 1 + D^2, \end{cases} \qquad \text{(EQ 18)}$$

respectively. Furthermore, we define

$$\begin{cases} B_0(D) = D^2 + D^3 + D^5, \\ B_1(D) = 1 + D + D^4, \end{cases} \tag{EQ 19}$$

which are used to create the length seven Barker sequences. Then, the access code is generated by the following procedure:

1. Format the 30 information bits to encode:

$$x(D) = a(D) + D^{24}B_{a_{23}}(D).$$

2. Add the information covering part of the PN overlay sequence:

$$\tilde{x}(D) = x(D) + p_{34} + p_{35}D + \ldots + p_{63}D^{29}.$$

3. Generate parity bits of the (64,30) expurgated block code:[1]

$$\tilde{c}(D) = D^{34}\tilde{x}(D) \bmod g(D).$$

4. Create the codeword:

$$\tilde{s}(D) = D^{34}\tilde{x}(D) + \tilde{c}(D).$$

5. Add the PN sequence:

$$s(D) = \tilde{s}(D) + p(D).$$

6. Append the (DC-free) preamble and trailer:

$$y(D) = F_{c_0}(D) + D^4 s(D) + D^{68}F_{a_{23}}(D).$$

13.2.2 Pseudo-random noise sequence generation

To generate the pseudo-random noise sequence we use the primitive polynomial $h(D) = 1 + D + D^3 + D^4 + D^6$. The LFSR and its starting state are shown in Figure 13.3 on page 147. The PN sequence generated (including the extra terminating zero) becomes (hexadecimal notation) 83848D96BBCC54FC. The LFSR output starts with the left-most bit of this PN sequence. This corresponds to $p'(D)$ of the previous section. Thus, using the reciprocal $p(D)$ as overlay gives the 64 bit sequence

$$p = 3F2A33DD69B121C1, \tag{EQ 20}$$

1. $x(D)$ mod $y(D)$ denotes the rest when $x(D)$ is divided by $y(D)$.

where the left-most bit is $p_0 = 0$ (there are two initial zeros in the binary representation of the hexadecimal digit 3), and $p_{63} = 1$ is the right-most bit.

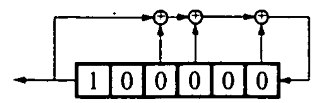


Figure 13.3: LFSR and the starting state to generate $p'(D)$

13.2.3 Reserved addresses for GIAC and DIAC

There is a block of 64 contiguous LAPs reserved for Bluetooth inquiry operations; one LAP common to all Bluetooth devices is reserved for general inquiry, the remaining 63 LAPs are reserved for dedicated inquiry of specific classes of Bluetooth devices. The same 64-block is used regardless of the contents of UAP and NAP. Consequently, none of these LAPs can be part of a user BD_ADDR.

When initializing HEC and CRC for the FHS packet of **inquiry response**, the UAP is replaced by DCI. Likewise, whenever one of the reserved BD_ADDRs is used for generating a frequency hop sequence, the UAP will be replaced by the DCI.

The reserved LAP addresses are tentatively chosen as `0x9E8B00`-`0x9E8B3F`. The general inquiry LAP is tentatively chosen to `0x9E8B33`. All addresses have the LSB at the rightmost position, hexadecimal notation.

13.3 ACTIVE MEMBER ADDRESS (AM_ADDR)

Each slave active in a piconet is assigned a 3-bit active member address (AM_ADDR). The all-zero AM_ADDR is reserved for broadcast messages. The master does not have an AM_ADDR. Its timing relative to the slaves distinguishes it from the slaves. A slave only accepts a packet with a matching AM_ADDR and broadcast packets. The AM_ADDR is carried in the packet header. The AM_ADDR is only valid as long as a slave is active on the channel. As soon as it is disconnected or parked, it loses the AM_ADDR.

The AM_ADDR is assigned by the master to the slave when the slave is activated. This is either at connection establishment or when the slave is unparked. At connection establishment, the AM_ADDR is carried in the **FHS** payload (the **FHS** header itself carries the all-zero AM_ADDR). When unparking, the AM_ADDR is carried in the unpark message.

13.4 PARKED MEMBER ADDRESS (PM_ADDR)

A slave in park mode can be identified by its BD_ADDR or by a dedicated parked member address (PM_ADDR). This latter address is a 8-bit member address that separates the parked slaves. The PM_ADDR is only valid as long as the slave is parked. When the slave is activated it is assigned an AM_ADDR but loses the PM_ADDR. The PM_ADDR is assigned to the slave the moment it is parked.

The all-zero PM_ADDR is reserved for parked slaves that only use their BD_ADDR to be unparked.

13.5 ACCESS REQUEST ADDRESS (AR_ADDR)

The access request address is used by the parked slave to determine the slave-to-master half slot in the access window it is allowed to send access request messages in, see also Section 10.8.4.6 on page 120. The AR_ADDR is assigned to the slave when it enters the park mode and is only valid as long as the slave is parked. The AR_ADDR is not necessarily unique; i.e. different parked slaves may have the same AR_ADDR.

14 BLUETOOTH SECURITY

The Bluetooth technology provides peer-to-peer communications over short distances. In order to provide usage protection and information confidentiality, the system has to provide security measures both at the application layer and the link layer. These measures shall be appropriate for a peer environment. This means that in each Bluetooth unit, the authentication and encryption routines are implemented in the same way. Four different entities are used for maintaining security at the link layer: a public address which is unique for each user[1], two secret keys, and a random number which is different for each new transaction. The four entities and their sizes as used in Bluetooth are summarized in Table 14.1.

Entity	Size
BD_ADDR	48 bits
Private user key, authentication	128 bits
Private user key, encryption configurable length (byte-wise)	8-128 bits
RAND	128 bits

Table 14.1: Entities used in authentication and encryption procedures.

The Bluetooth device address (BD_ADDR) is the 48-bit IEEE address which is unique for each Bluetooth unit. The Bluetooth addresses are publicly known, and can be obtained via MMI interactions, or, automatically, via an inquiry routine by a Bluetooth unit.

The secret keys are derived during initialization and are further never disclosed. Normally, the encryption key is derived from the authentication key during the authentication process. For the authentication algorithm, the size of the key used is always 128 bits. For the encryption algorithm, the key size may vary between 1 and 16 octets (8 - 128 bits). The size of the encryption key shall be configurable for two reasons. The first has to do with the many different requirements imposed on cryptographic algorithms in different countries – both w.r.t. export regulations and official attitudes towards privacy in general. The second reason is to facilitate a future upgrade path for the security without the need of a costly redesign of the algorithms and encryption hardware; increasing the effective key size is the simplest way to combat increased computing power at the opponent side. Currently (1999) it seems that an encryption key size of 64 bits gives satisfying protection for most applications.

The encryption key is entirely different from the authentication key (even though the latter is used when creating the former, as is described in Section 14.5.4 on page 177). Each time encryption is activated, a new encryption key

1. The BD_ADDR is not a secured identity.

shall be generated. Thus, the lifetime of the encryption key does not necessarily correspond to the lifetime of the authentication key.

It is anticipated that the authentication key will be more static in its nature than the encryption key – once established, the particular application running on the Bluetooth device decides when, or if, to change it. To underline the fundamental importance of the authentication key to a specific Bluetooth link, it will often be referred to as the link key.

The RAND is a random number which can be derived from a random or pseudo-random process in the Bluetooth unit. This is not a static parameter, it will change frequently.

In the remainder of this chapter, the terms user and application will be used interchangeably to designate the entity that is at the originating or receiving side.

14.1 RANDOM NUMBER GENERATION

Each Bluetooth unit has a random number generator. Random numbers are used for many purposes within the security functions – for instance, for the challenge-response scheme, for generating authentication and encryption keys, etc. Ideally, a true random generator based on some physical process with inherent randomness is used. Examples of such processes are thermal noise from a semiconductor or resistor and the frequency instability of a free running oscillator. For practical reasons, a software based solution with a pseudo-random generator is probably preferable. In general, it is quite difficult to classify the randomness of a pseudo-random sequence. Within Bluetooth, the requirements placed on the random numbers used are that they be non-repeating and randomly generated.

The expression 'non-repeating' means that it shall be highly unlikely that the value should repeat itself within the lifetime of the authentication key. For example, a non-repeating value could be the output of a counter that is unlikely to repeat during the lifetime of the authentication key, or a date/time stamp.

The expression 'randomly generated' means that it shall not be possible to predict its value with a chance that is significantly larger than 0 (e.g., greater than $1/2^L$ for a key length of L bits).

Clearly, the LM can use such a generator for various purposes; i.e. whenever a random number is needed (such as the RANDs, the unit keys, K_{init}, K_{master}, and random back-off or waiting intervals).

14.2 KEY MANAGEMENT

It is important that the encryption key size within a specific unit cannot be set by the user – this must be a factory preset entity. In order to prevent the user

from over-riding the permitted key size, the Bluetooth baseband processing does not accept an encryption key given from higher software layers. Whenever a new encryption key is required, it must be created as defined in Section 14.5.4 on page 177.

Changing a link key should also be done through the defined baseband procedures. Depending on what kind of link key it is, different approaches are required. The details are found in Section 14.2.2.7 on page 157.

14.2.1 Key types

The link key is a 128-bit random number which is shared between two or more parties and is the base for all security transactions between these parties. The link key itself is used in the authentication routine. Moreover, the link key is used as one of the parameters when the encryption key is derived.

In the following, a session is defined as the time interval for which the unit is a member of a particular piconet. Thus, the session terminates when the unit disconnects from the piconet.

The link keys are either semi-permanent or temporary. A semi-permanent link key is stored in non-volatile memory and may be used after the current session is terminated. Consequently, once a semi-permanent link key is defined, it may be used in the authentication of several subsequent connections between the Bluetooth units sharing it. The designation semi-permanent is justified by the possibility to change it. How to do this is described in Section 14.2.2.7 on page 157.

The lifetime of a temporary link key is limited by the lifetime of the current session – it cannot be reused in a later session. Typically, in a point-to-multipoint configuration where the same information is to be distributed securely to several recipients, a common encryption key is useful. To achieve this, a special link key (denoted master key) can temporarily replace the current link keys. The details of this procedure are found in Section 14.2.2.6 on page 157.

In the sequel we sometimes refer to what is denoted as the current link key. This is simply the link key in use at the current moment. It can be semi-permanent or temporary. Thus, the current link key is used for all authentications and all generation of encryption keys in the on-going connection (session).

In order to accommodate for different types of applications, four types of link keys have been defined:

- the combination key K_{AB}
- the unit key K_A
- the temporary key K_{master}
- the initialization key K_{init}

In addition to these keys there is an encryption key, denoted K_C. This key is derived from the current link key. Whenever the encryption is activated by a LM command, the encryption key shall be changed automatically. The purpose of separating the authentication key and encryption key is to facilitate the use of a shorter encryption key without weakening the strength of the authentication procedure. There are no governmental restrictions on the strength of authentication algorithms. However, in some countries, such restrictions exist on the strength of encryption algorithms.

For a Bluetooth unit, the combination key K_{AB} and the unit key K_A are functionally indistinguishable; the difference is in the way they are generated. The unit key K_A is generated in, and therefore dependent on, a single unit A. The unit key is generated once at installation of the Bluetooth unit; thereafter, it is very rarely changed. The combination key is derived from information in both units A and B, and is therefore always dependent on two units. The combination key is derived for each new combination of two Bluetooth units.

It depends on the application or the device whether a unit key or a combination key is used. Bluetooth units which have little memory to store keys, or, when installed in equipment that must be accessible to a large group of users, will preferably use their own unit key. In that case, they only have to store a single key. Applications that require a higher security level preferably use the combination keys. These applications will require more memory since a combination key for each link to a different Bluetooth unit has to be stored.

The master key, K_{master}, is a link key only used during the current session. It will replace the original link key only temporarily. For example, this may be utilized when a master wants to reach more than two Bluetooth units simultaneously using the same encryption key, see Section 14.2.2.6 on page 157.

The initialization key, K_{init}, is used as link key during the initialization process when no combination or unit keys have been defined and exchanged yet or when a link key has been lost. The initialization key protects the transfer of initialization parameters. The key is derived from a random number, an L-octet PIN code, and the BD_ADDR of the claimant unit. This key is only to be used during initialization.

The PIN can be a fixed number provided with the Bluetooth unit (for example when there is no MMI as in a PSTN plug). Alternatively, the PIN can be selected arbitrarily by the user, and then entered in both units that have to be matched. The latter procedure is used when both units have an MMI, for example a phone and a laptop. Entering a PIN in both units is more secure than using a fixed PIN in one of the units, and should be used whenever possible. Even if a fixed PIN is used, it shall be possible to change the PIN; this in order to prevent re-initialization by users who once got hold of the PIN. If no PIN is available, a default value of zero is to be used.

For many applications the PIN code will be a relatively short string of numbers.. Typically, it may consist of only four decimal digits. Even though this gives suffi-

cient security in many cases, there exist countless other, more sensitive, situations where this is not reliable enough. Therefore, the PIN code can be chosen to be any length from 1 to 16 octets. For the longer lengths, we envision the units exchanging PIN codes not through mechanical (i.e. human) interaction, but rather through means supported by software at the application layer. For example, this can be a Diffie-Hellman key agreement, where the exchanged key is passed on to the K_{init} generation process in both units, just as in the case of a shorter PIN code.

14.2.2 Key generation and initialization

The link keys have to be generated and distributed among the Bluetooth units in order to be used in the authentication procedure. Since the link keys must be secret, they cannot be obtained through an inquiry routine in the same way as the Bluetooth addresses. The exchange of the keys takes place during an initialization phase which has to be carried out separately for each two units that want to implement authentication and encryption. All initialization procedures consist of the following five parts:

- generation of an initialization key
- authentication
- generation of link key
- link key exchange
- generating of encryption key in each unit

After the initialization procedure, the units can proceed to communicate, or the link can be disconnected. If encryption is implemented, the E_0 algorithm is used with the proper encryption key derived from the current link key. For any new connection established between units A and B, they will use the common link key for authentication, instead of once more deriving K_{init} from the PIN. A new encryption key derived from that particular link key will be created next time encryption is activated.

If no link key is available, the LM shall automatically start an initialization procedure.

14.2.2.1 Generation of initialization key, K_{init}

A link key used temporarily during initialization is derived – the initialization key K_{init}. This key is derived by the E_{22} algorithm from the BD_ADDR of the claimant unit, a PIN code, the length of the PIN (in octets), and a random number IN_RAND_A issued (and created) by verifier. The principle is depicted in Figure 14.15 on page 177. The 128-bit output from E_{22} will be used for key exchange during the generation of a link key. It is also used for authentication when two

units have no record of a previous link key. When the units have performed the link key exchange, the initialization key shall be discarded.

When the initialization key is generated, the PIN is augmented with the BD_ADDR of the claimant unit. Since the maximum length of the PIN used in the algorithm cannot exceed 16 octets, it is possible that not all octets of BD_ADDR will be used. This procedure ensures that K_{init} depends on the identity of the unit trying to connect to it (at least when short PIN codes are used). A fraudulent Bluetooth unit may try to test a large number of PINs by each time claiming another BD_ADDR. It is the application's responsibility to take countermeasures against this threat. If the device address is kept fixed, the waiting interval until next try is permitted is increased exponentially (see Section 14.4.1 on page 170).

The details of the E_{22} algorithm can be found in Section 14.5.3 on page 175.

14.2.2.2 Authentication

The authentication procedure is carried out as described in Section 14.4 on page 169. If the two units have not been in contact before, the initialization key K_{init} is used as link key. Note that during each authentication, a new AU_RAND_A is issued.

Mutual authentication is achieved by first performing the authentication procedure in one direction and, if successful, immediately followed by performing the authentication procedure in the opposite direction.

As a side effect of a successful authentication procedure an auxiliary parameter, the Authenticated Ciphering Offset (ACO), will be computed. The ACO is used for ciphering key generation as described in Section 14.2.2.5 on page 156. In case of mutual authentication, the ACO value from the second authentication is retained. However, in some situations an authentication event may be initiated simultaneously in both devices. When this happens, there is no way of telling which is the first and which is the second event. Then, both units shall use the ACO resulting from the challenge generated in the master unit.

The claimant/verifier status is determined by the LM.

14.2.2.3 Generation of a unit key

A unit key K_A is generated when the Bluetooth unit is for the first time in operation; i.e. not during each initialization! The unit key is generated by the E_{21} algorithm as described in Section 14.5.3 on page 175. Once created, the unit key is stored in non-volatile memory and (almost) never changed. If after initialization the unit key is changed, the previously initialized units will possess a wrong link key. At initialization, the application has to determine which of the

two parties will provide the unit key as link key. Typically, this will be the unit with restricted memory capabilities, since this unit only has to remember its own unit key. The unit key is transferred to the other party and then stored as link key for that particular party. So, for example in Figure 14.1 on page 155, the unit key of unit A, K_A, is being used as link key for the connection A-B; unit A sends the unit key K_A to unit B; unit B will store K_A as the link key K_{BA}. For another initialization, for example with unit C, unit A will reuse its unit key K_A, whereas unit C stores it as K_{CA}.

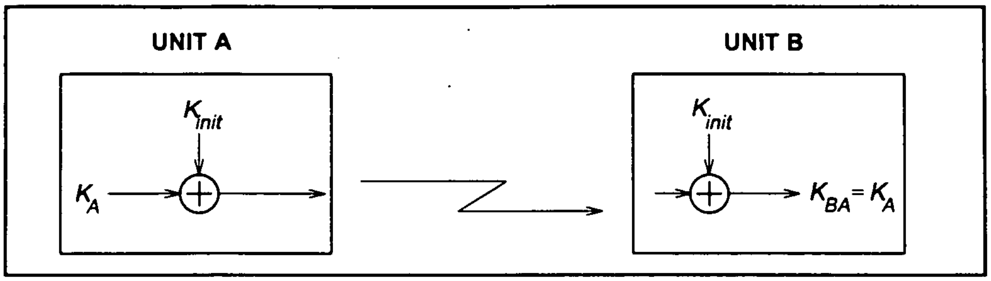


Figure 14.1: Generation of unit key. When the unit key has been exchanged, the initialization key shall be discarded in both units.

14.2.2.4 Generation of a combination key

If it is desired to use a combination key, this key is first generated during the initialization procedure. The combination key is the combination of two numbers generated in unit A and B, respectively. First, each unit generates a random number, say LK_RAND_A and LK_RAND_B. Then, utilizing E_{21} with the random number and the own BD_ADDR, the two random numbers

$$LK\_K_A = E_{21}(LK\_RAND_A, BD\_ADDR_A), \quad \text{(EQ 21)}$$

and

$$LK\_K_B = E_{21}(LK\_RAND_B, BD\_ADDR_B), \quad \text{(EQ 22)}$$

are created in unit A and unit B, respectively. These numbers constitute the units' contribution to the combination key that is to be created. Then, the two random numbers LK_RAND_A and LK_RAND_B are exchanged securely by XOR:ing with the current link key, say K. Thus, unit A sends $K \oplus LK\_RAND_A$ to unit B, while unit B sends $K \oplus LK\_RAND_B$ to unit A. Clearly, if this is done during the initialization phase the link key $K = K_{init}$.

When the random numbers LK_RAND_A and LK_RAND_B have been mutually exchanged, each unit recalculates the other units contribution to the combination key. This is possible since each unit knows the Bluetooth device address of the other unit. Thus, unit A calculates (EQ 22) and unit B calculates (EQ 21).

After this, both units combine the two numbers to generate the 128-bit link key. The combining operation is a simple bitwise modulo-2 addition (i.e. XOR). The result is stored in unit A as the link key K_{AB} and in unit B as the link key K_{BA}. When both units have derived the new combination key, a mutual authentication procedure shall be initiated to confirm the success of the transaction. The old link key shall be discarded after a successful exchange of a new combination key. The message flow between master and slave and the principle for creating the combination key is depicted in Figure 14.2 on page 156.

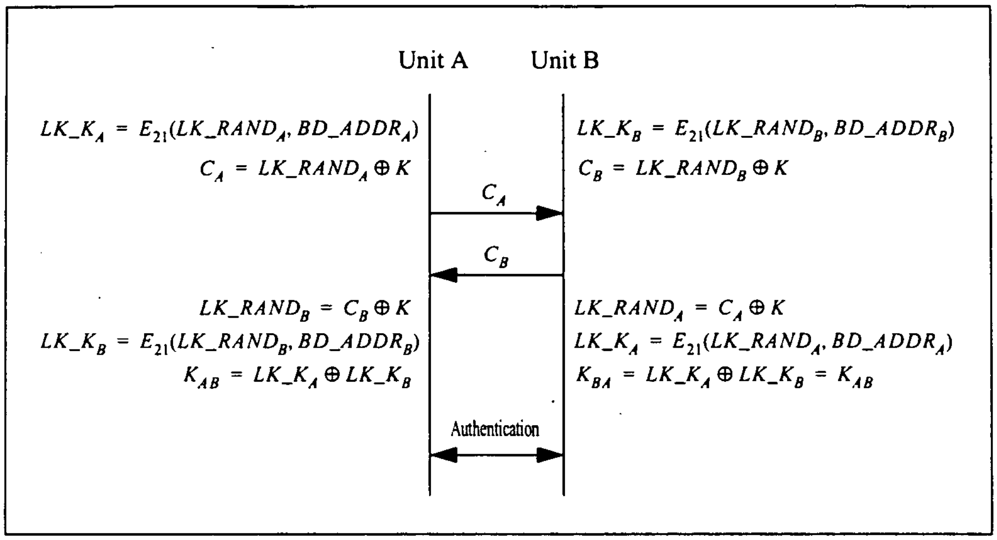


Figure 14.2: Generating a combination key. The old link key (K) shall be discarded after the exchange of a new combination key has succeeded

14.2.2.5 Generating the encryption key

The encryption key, K_C, is derived by algorithm E_3 from the current link key, a 96-bit Ciphering OFfset number (COF), and a 128-bit random number. The COF is determined in one of two ways. If the current link key is a master key, then COF is derived from the master BD_ADDR. Otherwise the value of COF is set to the value of ACO as computed during the authentication procedure. More precisely, we have[1]

$$\mathrm{COF} = \begin{cases} \mathrm{BD\_ADDR} \cup \mathrm{BD\_ADDR}, & \text{if link key is a master key} \\ \mathrm{ACO}, & \text{otherwise.} \end{cases} \qquad \text{(EQ 23)}$$

There is an explicit call of E_3 when the LM activates encryption. Consequently, the encryption key is automatically changed each time the unit enters the

1. $x \cup y$ denotes the concatenation of the octet strings x and y.

encryption mode. The details of the key generating function E_3 can be found in Section 14.5.4 on page 177.

14.2.2.6 Point-to-multipoint configuration

It is quite possible for the master to use separate encryption keys for each slave in a point-to-multipoint configuration with ciphering activated. Then, if the application requires more than one slave to listen to the same payload, each slave must be addressed individually. This may cause unwanted capacity loss for the piconet. Moreover, a Bluetooth unit (slave) is not capable of switching between two or more encryption keys in real time (e.g., after looking at the AM_ADDR in the header). Thus, the master cannot use different encryption keys for broadcast messages and individually addressed traffic. Alternatively, the master may tell several slave units to use a common link key (and, hence, indirectly also to use a common encryption key) and broadcast the information encrypted. For many applications, this key is only of temporary interest. In the sequel, this key is denoted K_{master}.

The transfer of necessary parameters is protected by the routine described in Section 14.2.2.8 on page 158. After the confirmation of successful reception in each slave, the master shall issue a command to the slaves to replace their respective current link key by the new (temporary) master key. Before encryption can be activated, the master also has to generate and distribute a common EN_RAND to all participating slaves. Using this random number and the newly derived master key, each slave generates a new encryption key.

Note that the master must negotiate what encryption key length to use individually with each slave who wants to use the master key. Since the master has already negotiated at least once with each slave, it has some knowledge of what sizes can be accepted by the different slaves. Clearly, there might be situations where the permitted key lengths of some units are incompatible. In that case, the master must have the limiting unit excluded from the group.

When all slaves have received the necessary data, the master can communicate information on the piconet securely using the encryption key derived from the new temporary link key. Clearly, each slave in possession of the master key can eavesdrop on all encrypted traffic, not only the traffic intended for itself. If so desired, the master can tell all participants to fall back to their old link keys simultaneously.

14.2.2.7 Modifying the link keys

In certain circumstances, it is desirable to be able to modify the link keys. A link key based on a unit key can be changed, but not very easily. The unit key is created once during the first use. Changing the unit key is a less desirable alternative, as several units may share the same unit key as link key (think of a printer whose unit key is distributed among all users using the printer's unit key

as link key). Changing the unit key will require re-initialization of all units trying to connect. In certain cases, this might be desirable; for example to deny access to previously allowed units.

If the key change concerns combination keys, then the procedure is rather straightforward. The change procedure can be identical to the procedure illustrated in Figure 14.2 on page 156, using the current value of the combination key as link key. This procedure can be carried out at any time after the authentication and encryption start. In fact, since the combination key corresponds to a single link, it can be modified each time this link is established. This will improve the security of the system since then old keys lose their validity after each session.

Of course, starting up an entirely new initialization procedure is also a possibility. In that case, user interaction is necessary since a PIN is required in the authentication and encryption procedures.

14.2.2.8 Generating a master key

The key-change routines described so far are semi-permanent. To create the master link key, which can replace the current link key during an initiated session (see Section 14.2.2.6), other means are needed. First, the master creates a new link key from two 128-bit random numbers, RAND1 and RAND2. This is done by

$$K_{master} = E_{22}(\mathrm{RAND1}, \mathrm{RAND2}, 16). \qquad \text{(EQ 24)}$$

Clearly, this key is a 128-bit random number. The reason to use the output of E_{22} and not directly chose a random number as the key, is to avoid possible problems with degraded randomness due to a poor implementation of the random number generator within the Bluetooth unit.

Then, a third random number, say RAND, is transmitted to the slave. Using E_{22} with the current link key and RAND as inputs, both the master and slave computes a 128-bit overlay. The master sends the bitwise XOR of the overlay and the new link key to the slave. The slave, who knows the overlay, recalculates K_{master}. To confirm the success of this transaction, the units can perform an authentication procedure using the new link key (with the master as verifier and the slave as claimant). This procedure is then repeated for each slave who shall receive the new link key. The ACO values from the involved authentications should not replace the current existing ACO as this ACO is needed to (re)compute a ciphering key when the master wants to fall back to the previous link (non-temporary) key.

When so required – and potentially long after the actual distribution of the master key – the master activates encryption by an LM command. Before doing that, the master must ensure that all slaves receive the same random number,

say EN_RAND, since the encryption key is derived through the means of E_3 individually in all participating units. Then, each slave computes a new encryption key,

$$K_C = E_3(K_{master}, \text{EN\_RAND}, \text{COF}), \quad \text{(EQ 25)}$$

where the value of COF is derived from the master's BD_ADDR as specified by equation (EQ 23). The details on the encryption key generating function can be found in Section 14.5.4 on page 177. The principle of the message flow between the master and slave when generating the master key is depicted in Figure 14.3. Note that in this case the ACO produced during the authentication is not used when computing the ciphering key.

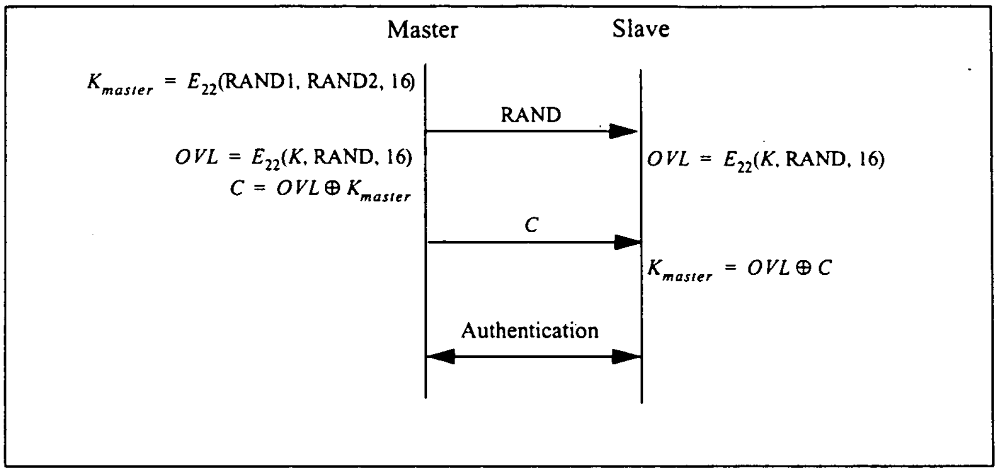


Figure 14.3: Master link key distribution and computation of the corresponding encryption key.

14.3 ENCRYPTION

User information can be protected by encryption of the packet payload; the access code and the packet header are never encrypted. The encryption of the payloads is carried out with a stream cipher called E_0 that is re-synchronized for every payload. The overall principle is shown in Figure 14.4 on page 160.

The stream cipher system E_0 consists of three parts. One part performs the initialization (generation of the payload key), the second part generates the key stream bits, and the third part performs the encryption and decryption. The payload key generator is very simple – it merely combines the input bits in an appropriate order and shift them into the four LFSRs used in the key stream generator. The main part of the cipher system is the second, as it also will be used for the initialization. The key stream bits are generated by a method derived from the summation stream cipher generator attributable to Massey and Rueppel. The method has been thoroughly investigated, and there exist good estimates of its strength with respect to presently known methods for cryptanalysis. Although the summation generator has weaknesses that can be

used in so-called correlation attacks, the high re-synchronization frequency will disrupt such attacks.

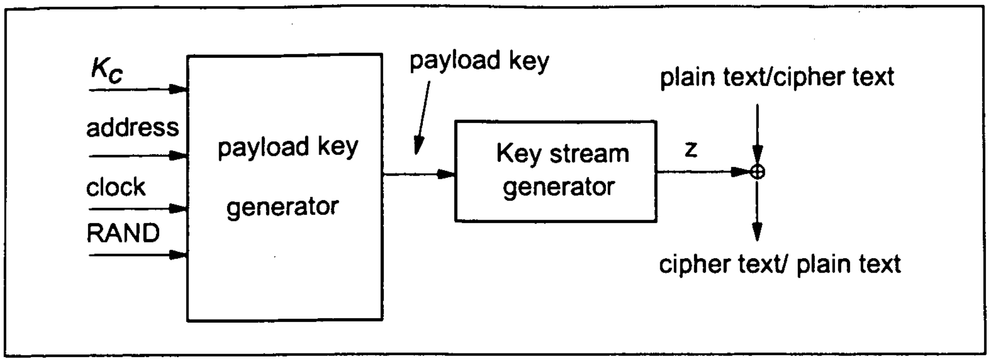


Figure 14.4: Stream ciphering for Bluetooth with E_0.

14.3.1 Encryption key size negotiation

Each Bluetooth device implementing the baseband specification needs a parameter defining the maximal allowed key length, L_{max}, $1 \leq L_{max} \leq 16$ (number of octets in the key). For each application, a number L_{min} is defined indicating the smallest acceptable key size for that particular application. Before generating the encryption key, the involved units must negotiate to decide what key size to actually use.

The master sends a suggested value, $L_{sug}^{(M)}$, to the slave. Initially, the suggested value is set to $L_{max}^{(M)}$. If $L_{min}^{(S)} \leq L_{sug}^{(M)}$, and, the slave supports the suggested length, the slave acknowledges and this value will be the length of the encryption key for this link. However, if both conditions are not fulfilled, the slave sends a new proposal, $L_{sug}^{(S)} < L_{sug}^{(M)}$, to the master. This value should be the largest among all supported lengths less than the previous master suggestion. Then, the master performs the corresponding test on the slave suggestion. This procedure is repeated until a key length agreement is reached, or, one unit aborts the negotiation. An abortion may be caused by lack of support for L_{sug} and all smaller key lengths, or if $L_{sug} < L_{min}$ in one of the units. In case of abortion Bluetooth link encryption can not be employed.

The possibility of a failure in setting up a secure link is an unavoidable consequence of letting the application decide whether to accept or reject a suggested key size. However, this is a necessary precaution. Otherwise a fraudulent unit could enforce a weak protection on a link by claiming a small maximum key size.

14.3.2 Encryption modes

If a slave has a semi-permanent link key (i.e. a combination key or a unit key), it can only accept encryption on slots individually addressed to itself (and, of course, in the reverse direction to the master). In particular, it will assume that broadcast messages are not encrypted. The possible traffic modes are described in Table 14.2. When an entry in the table refers to a link key, it means that the encryption/decryption engine uses the encryption key derived from that link key.

Broadcast traffic	Individually addressed traffic
No encryption	No encryption
No encryption	Encryption, Semi-permanent link key

Table 14.2: Possible traffic modes for a slave using a semi-permanent link key.

If the slave has received a master key, there are three possible combinations as defined in Table 14.3 on page 161. In this case, all units in the piconet uses a common link key, K_{master}. Since the master uses encryption keys derived from this link key for all secure traffic on the piconet, it is possible to avoid ambiguity in the participating slaves on which encryption key to use. Also in this case the default mode is that broadcast messages are not encrypted. A specific LM-command is required to activate encryption – both for broadcast and for individually addressed traffic.

Broadcast traffic	Individually addressed traffic
No encryption	No encryption
No encryption	Encryption, K_{master}
Encryption, K_{master}	Encryption, K_{master}

Table 14.3: Possible encryption modes for a slave in possession of a master key.

The master can issue an LM-command to the slaves telling them to fall back to their previous semi-permanent link key. Then, regardless of the previous mode they were in, they will end up in the first row of Table 14.2 on page 161; i.e. no encryption.

14.3.3 Encryption concept

For the encryption routine, a stream cipher algorithm will be used in which ciphering bits are bit-wise modulo-2 added to the data stream to be sent over the air interface. The payload is ciphered after the CRC bits are appended, but, prior to the FEC encoding.

Each packet payload is ciphered separately. The cipher algorithm E_0 uses the master Bluetooth address, 26 bits of the master realtime clock (CLK_{26-1}) and the encryption key K_C as input, see Figure 14.5 on page 162 (where it is assumed that unit A is the master).

The encryption key K_C is derived from the current link key, COF, and a random number, EN_RAND_A (see Section 14.5.4 on page 177). The random number is issued by the master before entering encryption mode. Note that EN_RAND_A is publicly known since it is transmitted as plain text over the air.

Within the E_0 algorithm, the encryption key K_C is modified into another key denoted K'_C . The maximum effective size of this key is factory preset and may be set to any multiple of eight between one and sixteen (8-128 bits). The procedure for deriving the key is described in Section 14.3.5 on page 165.

The real-time clock is incremented for each slot. The E_0 algorithm is re-initialized at the start of each new packet (i.e. for Master-to-Slave as well as for Slave-to-Master transmission). By using CLK_{26-1} at least one bit is changed between two transmissions. Thus, a new keystream is generated after each re-initialization. For packets covering more than a single slot, the Bluetooth clock as found in the first slot is being used for the entire packet.

The encryption algorithm E_0 generates a binary keystream, K_{cipher} , which is modulo-2 added to the data to be encrypted. The cipher is symmetric; decryption is performed in exactly the same way using the same key as used for encryption.

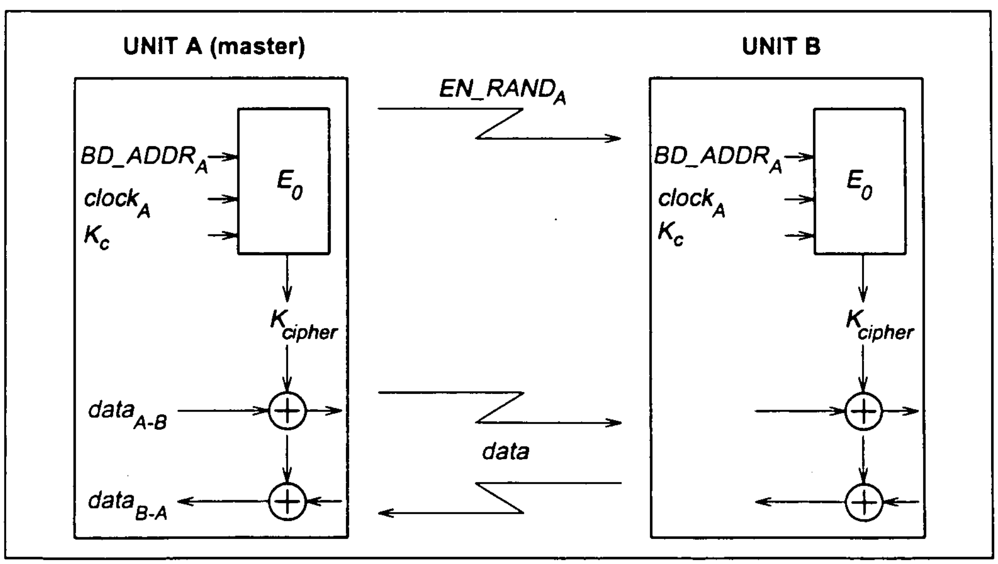


Figure 14.5: Functional description of the encryption procedure

14.3.4 Encryption algorithm

The system uses linear feedback shift registers (LFSRs) whose output is combined by a simple finite state machine (called the summation combiner) with 16 states. The output of this state machine is the key stream sequence, or, during initialization phase, the randomized initial start value. The algorithm is presented with an encryption key K_C, an 48-bit Bluetooth address, the master clock bits CLK_{26-1}, and a 128-bit RAND value. Figure 14.6 on page 163 shows the setup.

There are four LFSRs ($LFSR_1$,...,$LFSR_4$) of lengths $L_1 = 25$, $L_2 = 31$, $L_3 = 33$, and, $L_4 = 39$, with feedback polynomials as specified in Table 14.4 on page 164. The total length of the registers is 128. These polynomials are all primitive. The Hamming weight of all the feedback polynomials is chosen to be five – a reasonable trade-off between reducing the number of required XOR gates in the hardware realization and obtaining good statistical properties of the generated sequences.

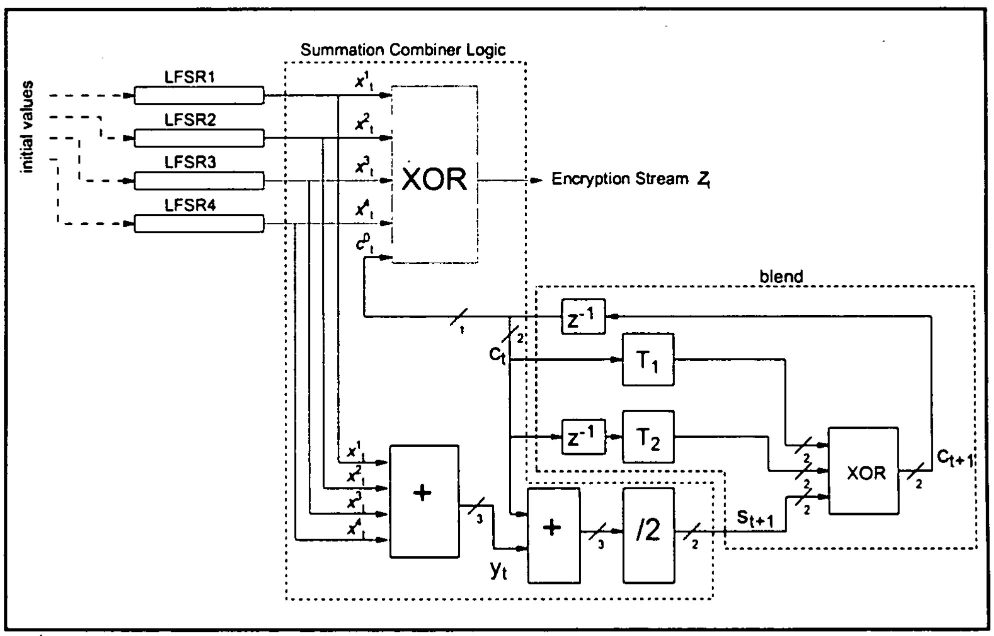


Figure 14.6: Concept of the encryption engine.

i	L_i	feedback $f_i(t)$	weight
1	25	$t^{25}+t^{20}+t^{12}+t^{8}+1$	5
2	31	$t^{31}+t^{24}+t^{16}+t^{12}+1$	5
3	33	$t^{33}+t^{28}+t^{24}+t^{4}+1$	5
4	39	$t^{39}+t^{36}+t^{28}+t^{4}+1$	5

Table 14.4: The four primitive feedback polynomials.

Let x_t^i denote the t^{th} symbol of LSFR$_i$. From the four-tuple $x_t^1, \ldots, x_t^4$ we derive the value y_t as

$$y_t = \sum_{i=1}^{4} x_t^i, \tag{EQ 26}$$

where the sum is over the integers. Thus y_t can take the values 0,1,2,3, or 4. The output of the summation generator is now given by the following equations

$$z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0 \in \{0, 1\}, \tag{EQ 27}$$

$$s_{t+1} = (s_{t+1}^1, s_{t+1}^0) = \left\lfloor \frac{y_t + c_t}{2} \right\rfloor \in \{0, 1, 2, 3\}, \tag{EQ 28}$$

$$c_{t+1} = (c_{t+1}^1, c_{t+1}^0) = s_{t+1} \oplus T_1[c_t] \oplus T_2[c_{t-1}], \tag{EQ 29}$$

where $T_1[.]$ and $T_2[.]$ are two different linear bijections over GF(4). Suppose GF(4) is generated by the irreducible polynomial $x^2 + x + 1$, and let α be a zero of this polynomial in GF(4). The mappings T_1 and T_2 are now defined as

$$\begin{aligned} T_1 &: \mathrm{GF}(4) \to \mathrm{GF}(4) \\ & x \mapsto x \\ T_2 &: \mathrm{GF}(4) \to \mathrm{GF}(4) \\ & x \mapsto (\alpha + 1)x. \end{aligned}$$

We can write the elements of GF(4) as binary vectors. This is summarized in Table 14.5.

Since the mappings are linear, we can realize them using XOR gates; i.e.

x	$T_1[x]$	$T_2[x]$
00	00	00
01	01	11
10	10	01
11	11	10

Table 14.5: The mappings T_1 and T_2.

$$T_1: \quad (x_1, x_0) \mapsto (x_1, x_0),$$
$$T_2: \quad (x_1, x_0) \mapsto (x_0, x_1 \oplus x_0).$$

14.3.4.1 The operation of the cipher

Figure 14.7 on page 165 gives an overview of the operation in time. The encryption algorithm shall run through the initialization phase before the start of transmitting or receiving a new packet. Thus, for multislot packets the cipher is initialized using the clock value of the first slot in the multislot sequence.

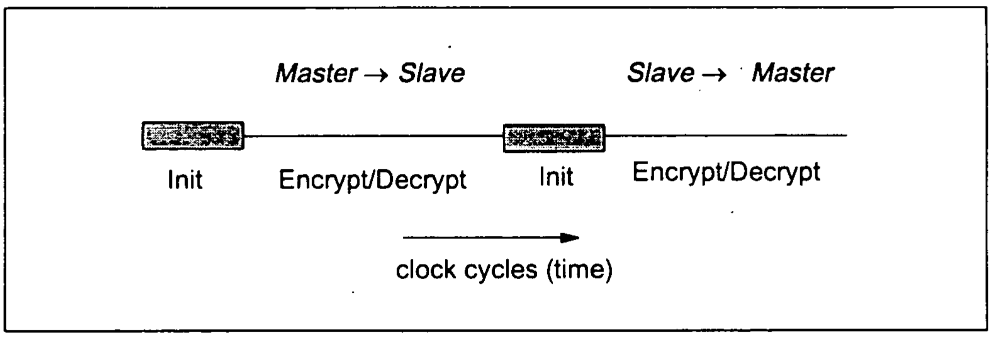


Figure 14.7: Overview of the operation of the encryption engine. Between each start of a packet (TX or RX), the LFSRs are re-initialized.

14.3.5 LFSR initialization

The key stream generator needs to be loaded with an initial value for the four LFSRs (in total 128 bits) and the 4 bits that specify the values of c_0 and c_{-1}. The 132 bit initial value is derived from four inputs by using the key stream generator itself. The input parameters are the key K_C, a 128-bit random number RAND, a 48-bit Bluetooth address, and the 26 master clock bits CLK_{26-1}.

The effective length of the encryption key can vary between 8 and 128 bits. Note that the actual key length as obtained from E_3 is 128 bits. Then, within E_0, the key length is reduced by a modulo operation between K_C and a polynomial of desired degree. After reduction, the result is encoded with a block code in

order to distribute the starting states more uniformly. The operation is defined in (EQ 30).

When the encryption key has been created the LFSRs are loaded with their initial values. Then, 200 stream cipher bits are created by operating the generator. Of these bits, the last 128 are fed back into the key stream generator as an initial value of the four LFSRs. The values of c_t and c_{t-1} are kept. From this point on, when clocked the generator produces the encryption (decryption) sequence which is bitwise XORed to the transmitted (received) payload data.

In the following, we will denote octet i of a binary sequence X by the notation $X[i]$. We define bit 0 of X to be the LSB. Then, the LSB of $X[i]$ corresponds to bit $8i$ of the sequence X, the MSB of $X[i]$ is bit $8i+7$ of X. For instance, bit 24 of the Bluetooth address is the LSB of ADR[3].

The details of the initialization are as follows:

1. Create the encryption key to use from the 128-bit secret key K_C and the 128-bit publicly known EN_RAND. Let L, $1 \le L \le 16$, be the effective key length in number of octets. The resulting encryption key will be denoted K'_C:

$$K'_C(x) = g_2^{(L)}(x)(K_C(x) \bmod g_1^{(L)}(x)), \qquad \text{(EQ 30)}$$

   where $\deg(g_1^{(L)}(x)) = 8L$ and $\deg(g_2^{(L)}(x)) \le 128 - 8L$. The polynomials are defined in Table 14.6.
2. Shift in the 3 inputs K'_C, the Bluetooth address, the clock, and the six-bit constant 111001 into the LFSRs. In total 208 bits are shifted in.
   a) Open all switches shown in Figure 14.8 on page 168;
   b) Arrange inputs bits as shown in Figure 14.8; Set the content of all shift register elements to zero. Set $t = 0$.
   c) Start shifting bits into the LFSRs. The rightmost bit at each level of Figure 14.8 is the first bit to enter the corresponding LFSR.
   d) When the first input bit at level i reaches the rightmost position of $LFSR_i$, close the switch of this LFSR.
   e) At $t = 39$ (when the switch of $LFSR_4$ is closed), reset both blend registers $c_{39} = c_{39-1} = 0$; Up to this point, the content of c_t and c_{t-1} has been of no concern. However, from this moment forward their content will be used in computing the output sequence.
   f) From now on output symbols are generated. The remaining input bits are continuously shifted into their corresponding shift register. When the last bit has been shifted in, the shift register is clocked with input = 0;

   Note: When finished, $LFSR_1$ has effectively clocked 30 times with feedback closed, $LFSR_2$ has clocked 24 times, $LFSR_3$ has

clocked 22 times, and $LFSR_4$ has effectively clocked 16 times with feedback closed.

3. To mix initial data, continue to clock until 200 symbols have been produced with all switches closed ($t = 239$);
4. Keep blend registers c_t and c_{t-1}, make a parallel load of the last 128 generated bits into the LFSRs according to Figure 14.9 at $t = 240$;

After the parallel load in item 4, the blend register contents will be updated for each subsequent clock.

L	deg	$g_1^{(L)}$	deg	$g_2^{(L)}$
1	[8]	00000000 00000000 00000000 0000011d	[119]	00e275a0 abd218d4 cf928b9b bf6cb08f
2	[16]	00000000 00000000 00000000 0001003f	[112]	0001e3f6 3d7659b3 7f18c258 cff6efef
3	[24]	00000000 00000000 00000000 010000db	[104]	000001be f66c6c3a b1030a5a 1919808b
4	[32]	00000000 00000000 00000001 000000af	[96]	00000001 6ab89969 de17467f d3736ad9
5	[40]	00000000 00000000 00000100 00000039	[88]	00000000 01630632 91da50ec 55715247
6	[48]	00000000 00000000 00010000 00000291	[77]	00000000 00002c93 52aa6cc0 54468311
7	[56]	00000000 00000000 01000000 00000095	[71]	00000000 000000b3 f7fffce2 79f3a073
8	[64]	00000000 00000001 00000000 0000001b	[63]	00000000 00000000 a1ab815b c7ec8025
9	[72]	00000000 00000100 00000000 00000609	[49]	00000000 00000000 0002c980 11d8b04d
10	[80]	00000000 00010000 00000000 00000215	[42]	00000000 00000000 0000058e 24f9a4bb
11	[88]	00000000 01000000 00000000 0000013b	[35]	00000000 00000000 0000000c a76024d7
12	[96]	00000001 00000000 00000000 000000dd	[28]	00000000 00000000 00000000 1c9c26b9
13	[104]	00000100 00000000 00000000 0000049d	[21]	00000000 00000000 00000000 0026d9e3
14	[112]	00010000 00000000 00000000 0000014f	[14]	00000000 00000000 00000000 00004377
15	[120]	01000000 00000000 00000000 000000e7	[7]	00000000 00000000 00000000 00000089
16	[128]	1 00000000 00000000 00000000 00000000	[0]	00000000 00000000 00000000 00000001

Table 14.6: Polynomials used when creating K'_C.
All polynomials are in hexadecimal notation. The LSB is in the rightmost position.

In Figure 14.8, all bits are shifted into the LFSRs, starting with the least significant bit (LSB). For instance, from the third octet of the address, ADR[2], first ADR_{16} is entered, followed by ADR_{17}, etc. Furthermore, CL_0 corresponds to CLK_1,..., CL_{25} corresponds to CLK_{26}.

Note that the output symbols x_t^i, $i = 1, \ldots, 4$ are taken from the positions 24, 24, 32, and 32 for $LFSR_1$, $LFSR_2$,$LFSR_3$, and $LFSR_4$, respectively (counting the leftmost position as number 1).

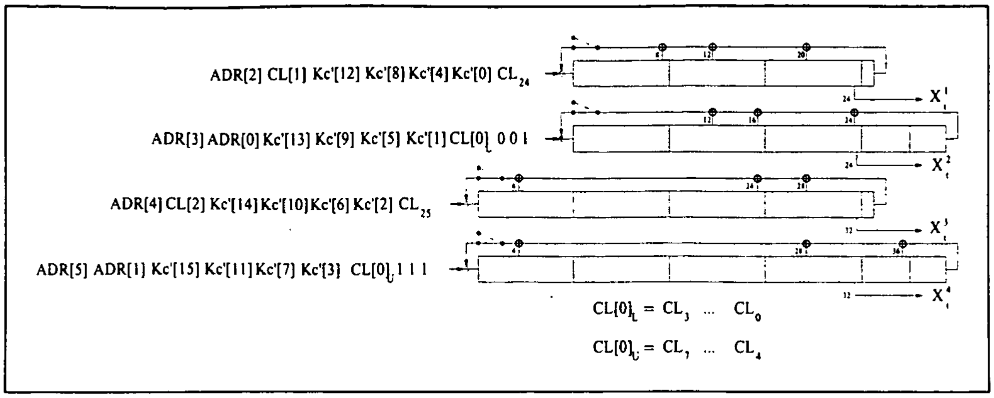


Figure 14.8: Arranging the input to the LFSRs.

In Figure 14.9, the 128 binary output symbols $Z_0, \ldots, Z_{127}$ are arranged in octets denoted Z[0],..., Z[15]. The LSB of Z[0] corresponds to the first of these symbols, the MSB of Z[15] is the latest output from the generator. These bits shall be loaded into the LFSRs according to the figure. It is a parallel load and no update of the blend registers is done. The first output symbol is generated at the same time. The octets are written into the registers with the LSB in the left-most position (i.e. the opposite of before). For example, Z_{24} is loaded into position 1 of $LFSR_4$.

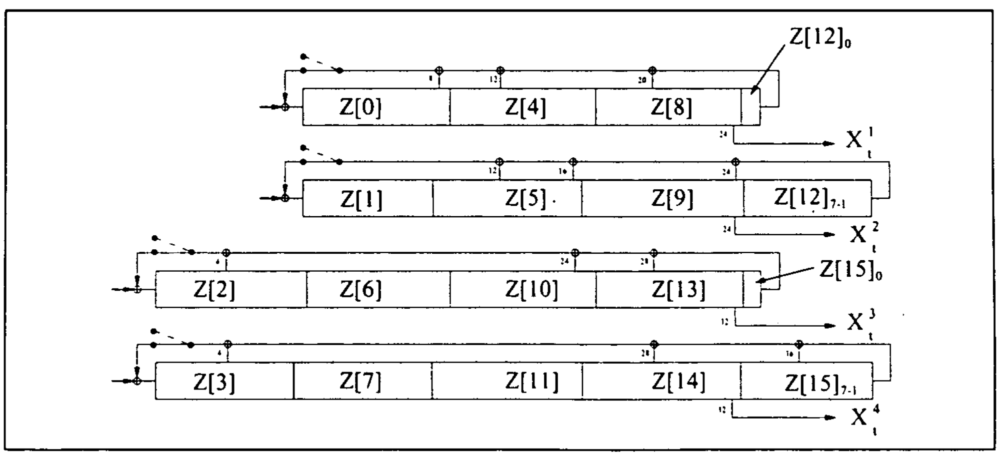


Figure 14.9: Distribution of the 128 last generated output symbols within the LFSRs.

14.3.6 Key stream sequence

When the initialization is finished, the output from the summation combiner is used for encryption/decryption. The first bit to use is the one produced at the parallel load, i.e. at $t = 240$. The circuit is run for the entire length of the current payload. Then, before the reverse direction is started, the entire initialization process is repeated with updated values on the input parameters.

Sample data of the encryption output sequence can be found in "Appendix IV" on page 899, Encryption Sample Data. A necessary, but not sufficient, condition for all Bluetooth-compliant implementations is to produce these encryption streams for identical initialization values.

14.4 AUTHENTICATION

The entity authentication used in Bluetooth uses a challenge-response scheme in which a claimant's knowledge of a secret key is checked through a 2-move protocol using symmetric secret keys. The latter implies that a correct claimant/verifier pair share the same secret key, for example K. In the challenge-response scheme the verifier challenges the claimant to authenticate a random input (the challenge), denoted by AU_RAND_A, with an authentication code, denoted by E_1, and return the result SRES to the verifier, see Figure 14.10 on page 169. This figure shows also that in Bluetooth the input to E_1 consists of the tuple AU_RAND_A and the Bluetooth device address (BD_ADDR) of the claimant. The use of this address prevents a simple reflection attack[1]. The secret K shared by units A and B is the current link key.

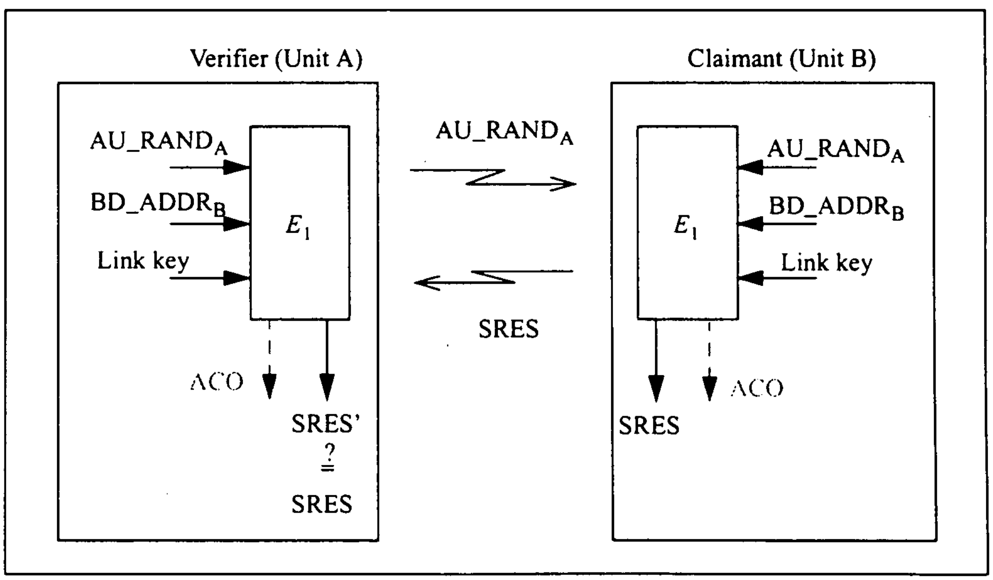


Figure 14.10: Challenge-response for the Bluetooth.

The challenge-response scheme for symmetric keys used in the Bluetooth is depicted in Figure 14.11 on page 170.

1. The reflection attack actually forms no threat in Bluetooth because all service requests are dealt with on a FIFO bases. When pre-emption is introduced, this attack is potentially dangerous.

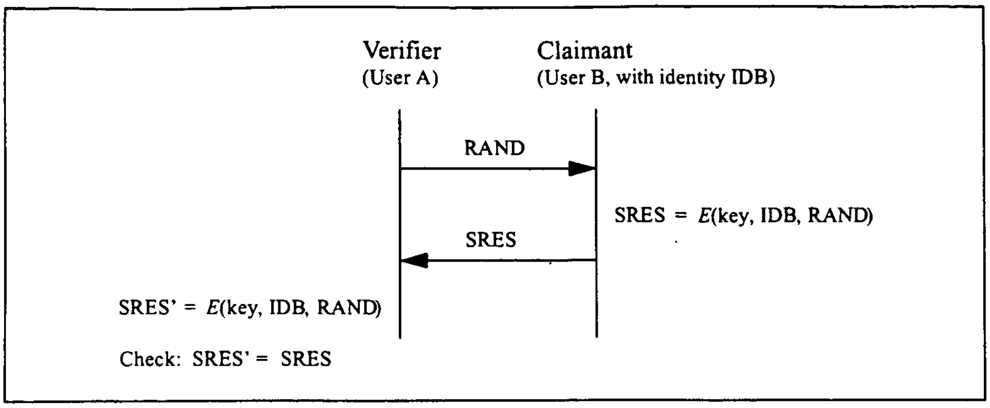


Figure 14.11: Challenge-response for symmetric key systems.

In the Bluetooth, the verifier is not necessarily the master. The application indicates who has to be authenticated by whom. Certain applications only require a one-way authentication. However, in some peer-to-peer communications, one might prefer a mutual authentication in which each unit is subsequently the challenger (verifier) in two authentication procedures. The LM coordinates the indicated authentication preferences by the application to determine in which direction(s) the authentication(s) has to take place. For mutual authentication with the units of Figure 14.10 on page 169, after unit A has successfully authenticated unit B, unit B could authenticate unit A by sending a AU_RAND_B (different from the AU_RAND_A that unit A issued) to unit A, and deriving the SRES and SRES' from the new AU_RAND_B, the address of unit A, and the link key K_{AB}.

If an authentication is successful the value of ACO as produced by E_1 should be retained.

14.4.1 Repeated attempts

When the authentication attempt fails, a certain waiting interval must pass before a new authentication attempt can be made. For each subsequent authentication failure with the same Bluetooth address, the waiting interval shall be increased exponentially. That is, after each failure, the waiting interval before a new attempt can be made, for example, twice as long as the waiting interval prior to the previous attempt[1]. The waiting interval shall be limited to a maximum. The maximum waiting interval depends on the implementation. The waiting time shall exponentially decrease to a minimum when no new failed attempts are being made during a certain time period. This procedure prevents an intruder to repeat the authentication procedure with a large number of different keys.

1. An other appropriate value larger than 1 may be used.

To make the system somewhat less vulnerable to denial-of-service attacks, the Bluetooth units should keep a list of individual waiting intervals for each unit it has established contact with. Clearly, the size of this list must be restricted only to contain the N units with which the most recent contact has been made. The number N can vary for different units depending on available memory size and user environment.

14.5 THE AUTHENTICATION AND KEY-GENERATING FUNCTIONS

This section describes the algorithmic means for supporting the Bluetooth security requirements on authentication and key generation.

14.5.1 The authentication function E_1

The authentication function E_1 proposed for the Bluetooth is a computationally secure authentication code, or often called a MAC. E_1 uses the encryption function called SAFER+. The algorithm is an enhanced version[1] of an existing 64-bit block cipher SAFER-SK128, and it is freely available. In the sequel the block cipher will be denoted as the function A_r which maps under a 128-bit key, a 128-bit input to a 128-bit output, i.e.

$$\begin{aligned} A_r\colon\ & \{0,1\}^{128} \times \{0,1\}^{128} \rightarrow \{0,1\}^{128} \\ & (k \times x) \mapsto t. \end{aligned} \tag{EQ 31}$$

The details of A_r are given in the next section. The function E_1 is constructed using A_r as follows

$$\begin{aligned} E_1\colon\ & \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{48} \rightarrow \{0,1\}^{32} \times \{0,1\}^{96} \\ & (K, \mathrm{RAND}, \mathrm{address}) \mapsto (\mathrm{SRES}, \mathrm{ACO}), \end{aligned} \tag{EQ 32}$$

where $\mathrm{SRES} = Hash(K, \mathrm{RAND}, \mathrm{address}, 6)[0, \ldots, 3]$, where $Hash$ is a keyed hash function defined as[2],

$$\begin{aligned} & Hash\colon \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{8 \times L} \times \{6, 12\} \rightarrow \{0,1\}^{128} \\ & (K, I_1, I_2, L) \mapsto A'_r([\tilde{K}], [E(I_2, L) +_{16} (A_r(K, I_1) \oplus_{16} I_1)]), \end{aligned} \tag{EQ 33}$$

and where

1. It is presently one of the contenders for the Advanced Encryption Standard (AES) submitted by Cylink, Corp, Sunnyvale, USA
2. The operator $+_{16}$ denotes bytewise addition mod 256 of the 16 octets, and the operator $\oplus_{16}$ denotes bytewise XORing of the 16 octets.

$$E:\{0,1\}^{8\times L}\times\{6,12\}\to\{0,1\}^{8\times 16}$$
$$(X[0,\ldots,L-1],L)\mapsto(X[i(\mathrm{mod}\ \mathrm{L})]\ \ \text{for } i=0\ldots15),\qquad\text{(EQ 34)}$$

is an expansion of the L octet word X into a 128-bit word. Thus we see that we have to evaluate the function A_r twice for each evaluation of E_1. The key $\tilde{K}$ for the second use of A_r (actually A'_r) is offseted from K as follows[1]

$$\begin{aligned}
&\tilde{K}[0]=(K[0]+233)\ \mathrm{mod}\ 256, && \tilde{K}[1]=K[1]\oplus 229,\\
&\tilde{K}[2]=(K[2]+223)\ \mathrm{mod}\ 256, && \tilde{K}[3]=K[3]\oplus 193,\\
&\tilde{K}[4]=(K[4]+179)\ \mathrm{mod}\ 256, && \tilde{K}[5]=K[5]\oplus 167,\\
&\tilde{K}[6]=(K[6]+149)\ \mathrm{mod}\ 256, && \tilde{K}[7]=K[7]\oplus 131,\\
&\tilde{K}[8]=K[8]\oplus 233, && \tilde{K}[9]=(K[9]+229)\ \mathrm{mod}\ 256,\\
&\tilde{K}[10]=K[10]\oplus 223, && \tilde{K}[11]=(K[11]+193)\ \mathrm{mod}\ 256,\\
&\tilde{K}[12]=K[12]\oplus 179, && \tilde{K}[13]=(K[13]+167)\ \mathrm{mod}\ 256,\\
&\tilde{K}[14]=K[14]\oplus 149, && \tilde{K}[15]=(K[15]+131)\ \mathrm{mod}\ 256.
\end{aligned}\qquad\text{(EQ 35)}$$

A data flowchart of the computation of E_1 is depicted in Figure 14.12 on page 173. E_1 is also used to deliver the parameter ACO (Authenticated Ciphering Offset) that is used in the generation of the ciphering key by E_3, see equations (EQ 23) and (EQ 43). The value of ACO is formed by the octets 4 through 15 of the output of the hash function defined in (EQ 33), i.e.

$$\mathrm{ACO}=\mathit{Hash}(K,\mathrm{RAND},\mathrm{address},6)[4,\ldots,15].\qquad\text{(EQ 36)}$$

1. The constants are the first largest primes below 257 for which 10 is a primitive root.

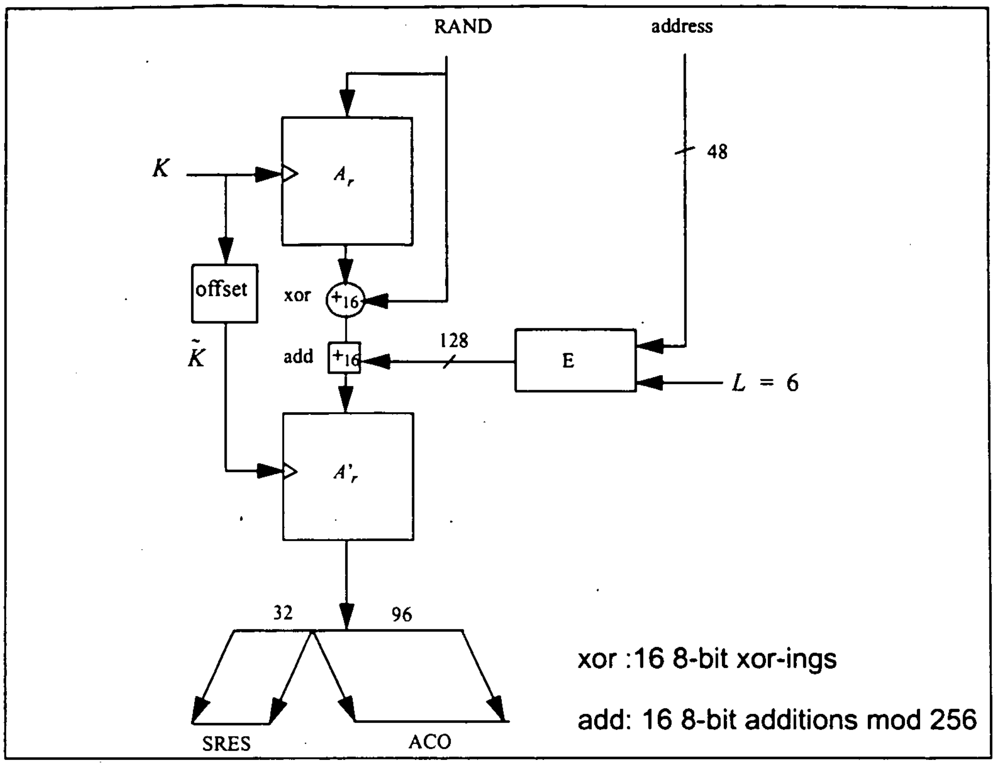


Figure 14.12: Flow of data for the computation of E_1.

14.5.2 The functions A_r and A'_r

The function A_r is identical to SAFER+. It consists of a set of 8 layers, (each layer is called a round) and a parallel mechanism for generating the sub keys $K_p[j]$, $p = 1, 2, \ldots, 17$, the so-called round keys to be used in each round. The function will produce a 128-bit result from a 128-bit "random" input string and a 128-bit "key". Besides the function A_r, a slightly modified version referred to as A'_r is used in which the input of round 1 is added to the input of the 3rd round. This is done to make the modified version non-invertible and prevents the use of A'_r (especially in E_{2x}) as an encryption function. See Figure 14.13 on page 174 for details.

14.5.2.1 The round computations

The computations in each round are a composition of encryption with a round key, substitution, encryption with the next round key, and, finally, a Pseudo Hadamard Transform (PHT). The computations in a round are shown inFigure 14.13 on page 174. The sub keys for round r, $r = 1, 2, \ldots, 8$ are denoted

$K_{2r-1}[j]$, $K_{2r}[j]$, $j = 0, 1, \ldots, 15$. After the last round $k_{17}[j]$ is applied in a similar fashion as all previous odd numbered keys.

14.5.2.2 The substitution boxes "e" and "l"

In Figure 14.13 on page 174 two boxes occur, marked "e" and "l". These boxes implement the same substitutions as used in SAFER+; i.e. they implement

$$\begin{array}{lll} e, l & : & \{0, \ldots, 255\} \rightarrow \{0, \ldots, 255\}, \\ e & : & i \mapsto (45^i \pmod{257}) \pmod{256}, \\ l & : & i \mapsto j \text{ s.t. } i = e(j). \end{array}$$

Their role, as in the SAFER+ algorithm, is to introduce non-linearity.

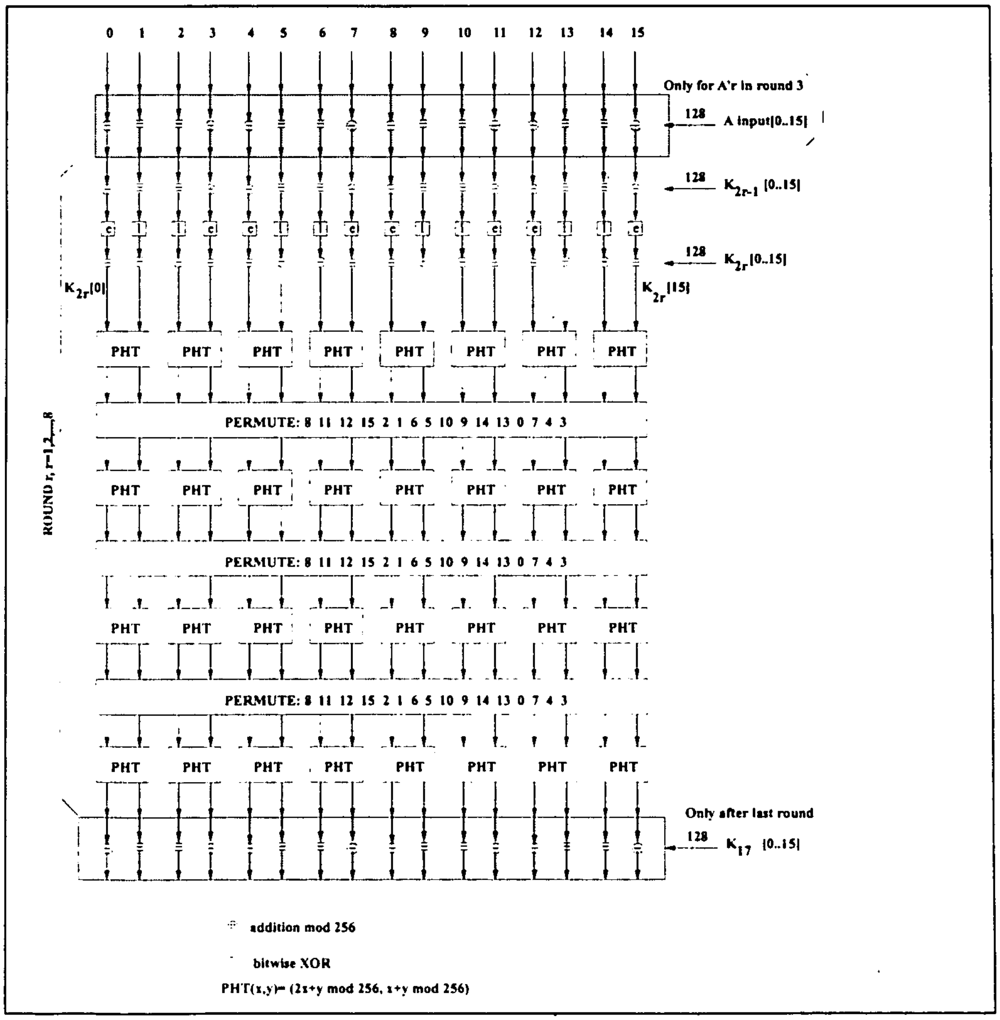


Figure 14.13: One round in A_r and A'_r. The permuatation boxes show how input byte indices are mapped onto output byte indices. Thus, position 0 (leftmost) is mapped on position 8, position 1 is mapped on position 11, et cetera.

14.5.2.3 Key scheduling

In each round, 2 batches of 16 octet-wide keys are needed. These so-called round keys are derived as specified by the key scheduling in SAFER+. Figure 14.14 on page 175 gives an overview of how the round keys $K_p[j]$ are determined. The bias vectors B_2, B_3, ..., B_{17} are computed according to following equation:

$$B_p[i] = ((45^{(45^{17p+i+1} \bmod 257)} \bmod 257) \bmod 256), \text{ for } i = 0, \ldots, 15. \qquad \text{(EQ 37)}$$

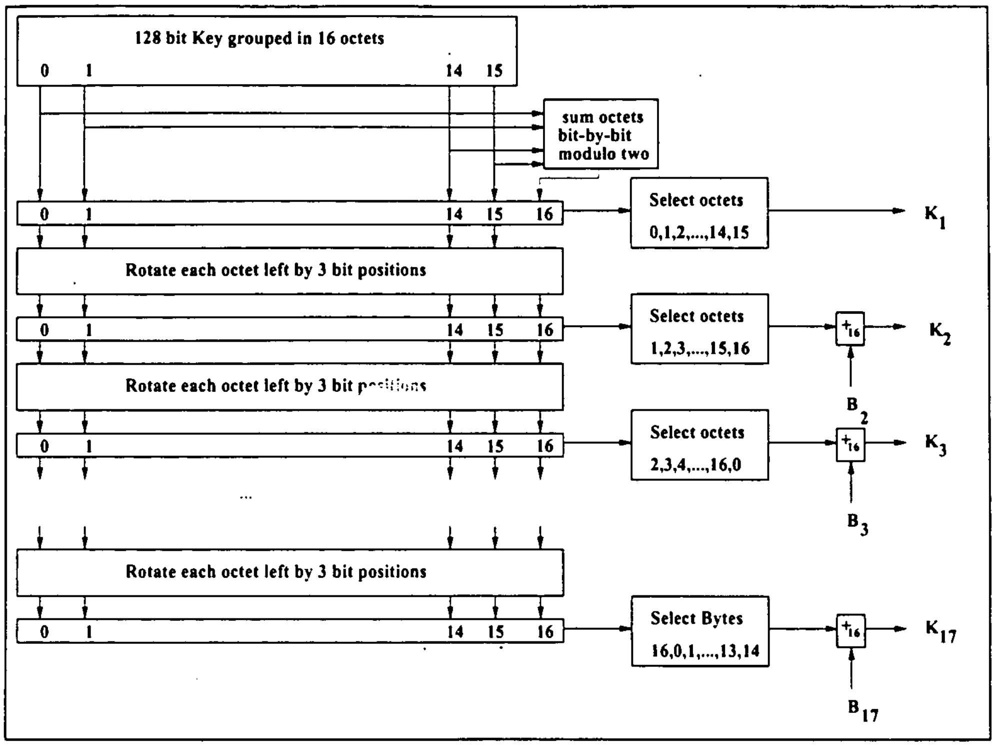


Figure 14.14: Key scheduling in A_r.

14.5.3 E_2-Key generation function for authentication

The key used for authentication is derived through a procedure that is shown in Figure 14.15 on page 177. The figure shows two different modes of operation for the algorithm. In the first mode, the function E_2 should produce on input of a 128-bit RAND value and a 48-bit address, a 128-bit link key K. This mode is utilized when creating unit keys and combination keys. In the second mode the function E_2 should produce, on input of a 128-bit RAND value and an L octet user PIN, a 128-bit link key K. The second mode is used to create the initialization key, and also whenever a master key is to be generated.

When the initialization key is generated, the PIN is augmented with the BD_ADDR of the claimant unit. The augmentation always starts with the least significant octet of the address immediately following the most significant octet of the PIN. Since the maximum length of the PIN used in the algorithm cannot exceed 16 octets, it is possible that not all octets of BD_ADDR will be used.

This key generating algorithm again exploits the cryptographic function. Formally E_2 can be expressed for mode 1 (denoted E_{21}) as

$$\begin{aligned} E_{21}&: \{0,1\}^{128} \times \{0,1\}^{48} \to \{0,1\}^{128} \\ &(\text{RAND}, \text{address}) \mapsto A'_r(X, Y) \end{aligned} \qquad \text{(EQ 38)}$$

where (for mode 1)

$$\begin{cases} X = \text{RAND}[0\ldots14] \cup (\text{RAND}[15] \oplus 6) \\ Y = \bigcup_{i=0}^{15} \text{address}[i \pmod 6] \end{cases} \qquad \text{(EQ 39)}$$

Let L be the number of octets in the user PIN. The augmenting is defined by

$$\text{PIN}' = \begin{cases} \text{PIN}[0\ldots L-1] \cup \text{BD\_ADDR}_B[0\ldots\min\{5, 15-L\}], & L < 16, \\ \text{PIN}[0\ldots L-1], & L = 16, \end{cases} \qquad \text{(EQ 40)}$$

where it is assumed that unit B is the claimant. Then, in mode 2, E_2 (denoted E_{22}) can be expressed as

$$\begin{aligned} E_{22}&: \{0,1\}^{8L'} \times \{0,1\}^{128} \times \{1, 2, \ldots, 16\} \to \{0,1\}^{128} \\ &(\text{PIN}', \text{RAND}, L') \mapsto A'_r(X, Y) \end{aligned} \qquad \text{(EQ 41)}$$

where

$$\begin{cases} X = \bigcup_{i=0}^{15} \text{PIN}'[i \pmod{L'}], \\ Y = \text{RAND}[0\ldots14] \cup (\text{RAND}[15] \oplus L'), \end{cases} \qquad \text{(EQ 42)}$$

and $L' = \min\{16, L+6\}$ is the number of octets in PIN'.

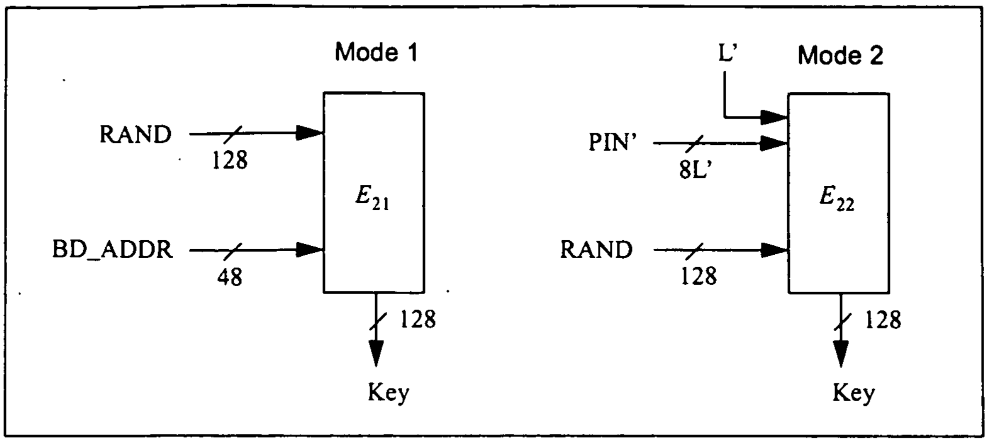


Figure 14.15: Key generating algorithm E_2 and its two modes. Mode 1 is used for unit and combination keys, while mode 2 is used for K_{init} and K_{master}.

14.5.4 E_3-Key generation function for encryption

The ciphering key K_C used by E_0 is generated by E_3. The function E_3 is constructed using A'_r as follows

$$E_3\colon \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{96} \to \{0,1\}^{128}$$
$$(K, \text{RAND}, \text{COF}) \mapsto Hash(K, \text{RAND}, \text{COF}, 12) \qquad \text{(EQ 43)}$$

where $Hash$ is the hash function as defined by (EQ 33). Note that the produced key length is 128 bits. However, before use within E_0, the encryption key K_C will be shortened to the correct encryption key length, as described in Section 14.3.5 on page 165. A block scheme of E_3 is depicted in Figure 14.16.

The value of COF is determined as specified by equation (EQ 23).

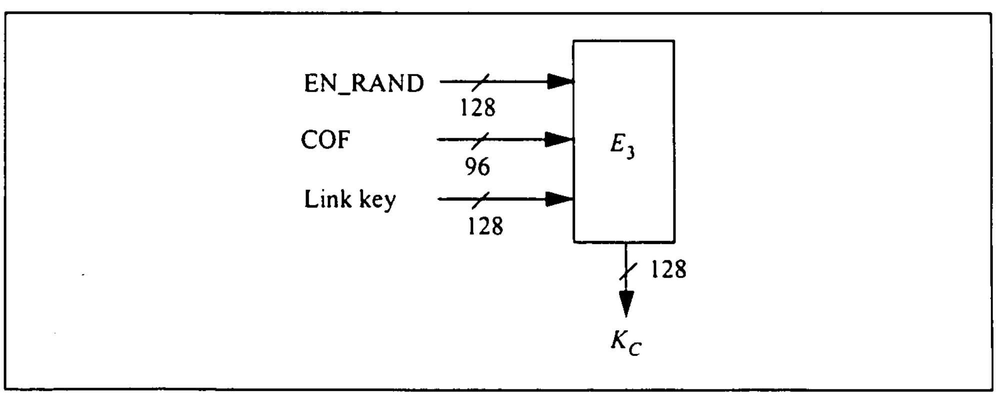


Figure 14.16: Generation of the encryption key.

15 LIST OF FIGURES

16 LIST OF TABLES

Part C

LINK MANAGER PROTOCOL

This specification describes the Link Manager Protocol (LMP) which is used for link set-up and control. The signals are interpreted and filtered out by the Link Manager on the receiving side and are not propagated to higher layers.

CONTENTS

1 GENERAL

LMP messages are used for link set-up, security and control. They are transferred in the payload instead of L2CAP and are distinguished by a reserved value in the L_CH field of the payload header. The messages are filtered out and interpreted by LM on the receiving side and are not propagated to higher layers.

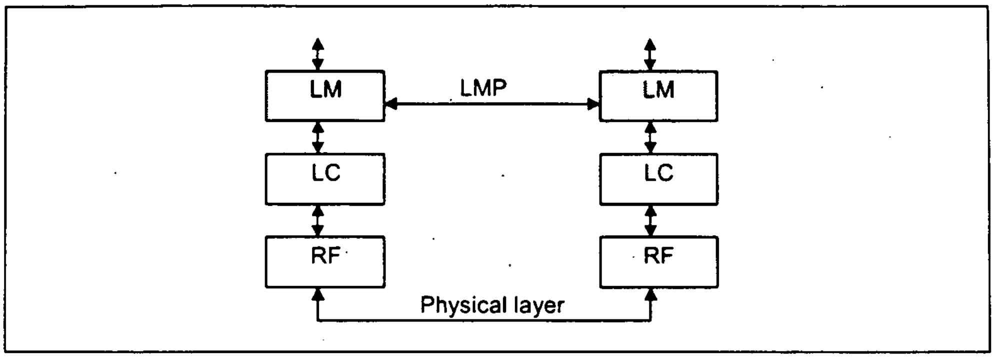


Figure 1.1: Link Manager's place on the global scene.

Link Manager messages have higher priority than user data. This means that if the Link Manager needs to send a message, it shall not be delayed by the L2CAP traffic, although it can be delayed by many retransmissions of individual baseband packets.

We do not need to explicitly acknowledge the messages in LMP since LC (see Baseband Specification Section 5, on page 67) provides us with a reliable link.

The time between receiving a baseband packet carrying an LMP PDU and sending a baseband packet carrying a valid response PDU, according to the procedure rules in Section 3 on page 193, must be less than the LMP Response Timeout. The value of this timeout is 30 seconds.

2 FORMAT OF LMP

LM PDUs are always sent as single-slot packets and the payload header is therefore one byte. The two least significant bits in the payload header determine the logical channel. For LM PDUs these bits are set.

L_CH code	Logical Channel	Information
00	NA	undefined
01	UA/I	Continuing L2CAP message
10	UA/I	Start L2CAP message
11	LM	LMP message

Table 2.1: Logical channel L_CH field contents.

The FLOW bit in the payload header is always one and is ignored on the receiving side. Each PDU is assigned a 7-bit opcode used to uniquely identify different types of PDUs, see Table 5.1 on page 226. The opcode and a one-bit transaction ID are positioned in the first byte of the payload body. The transaction ID is positioned in the LSB. It is 0 if the PDU belongs to a transaction initiated by the master and 1 if the PDU belongs to a transaction initiated by the slave. If the PDU contains one or more parameters these are placed in the payload starting at the second byte of the payload body. The number of bytes used depends on the length of the parameters. If an SCO link is present using HV1 packets and length of *content* is less than 9 bytes the PDUs can be transmitted in DV packets. Otherwise DM1 packets must be used. All parameters have little endian format, i.e. the least significant byte is transmitted first.

The source/destination of the PDUs is determined by the AM_ADDR in the packet header.

LSB	MSB
OpCode and transaction Id	Content

Figure 2.1: Payload body when LM PDUs are sent.

Each PDU is either mandatory or optional. The M/O field in the tables of Section 3 indicates this. The LM does not need to be able to transmit a PDU that is optional. The LM must recognize all optional PDUs that it receives and, if a response is required, send a valid response according to the procedure rules in Section 3. The reason that should be used in this case is *unsupported LMP feature.* If the optional PDU that is received does not require a response, no response is sent. Which of the optional PDUs a device supports can be requested, see Section 3.11 on page 205.

3 THE PROCEDURE RULES AND PDUs

Each procedure is described and depicted with a sequence diagram. The following symbols are used in the sequence diagrams:

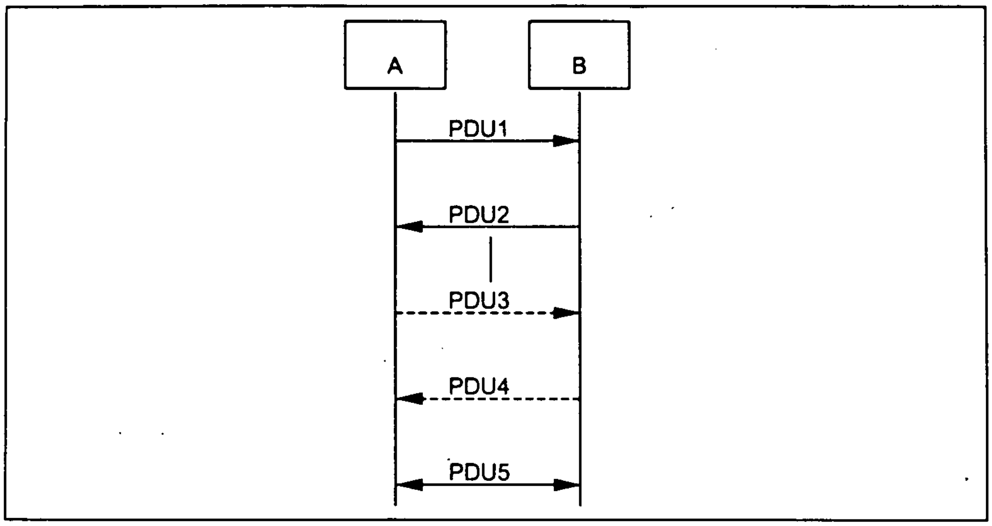


Figure 3.1: Symbols used in sequence diagrams.

PDU1 is a PDU sent from A to B. PDU2 is a PDU sent from B to A. PDU3 is a PDU that is optionally sent from A to B. PDU4 is a PDU that is optionally sent from B to A. PDU5 is a PDU sent from either A or B. A vertical line indicates that more PDUs can optionally be sent.

3.1 GENERAL RESPONSE MESSAGES

The PDUs LMP_accepted and LMP_not_accepted are used as response messages to other PDUs in a number of different procedures. The PDU LMP_accepted includes the opcode of the message that is accepted. The PDU LMP_not_accepted includes the opcode of the message that is not accepted and the reason why it is not accepted.

M/O	PDU	Contents
M	LMP_accepted	op code
M	LMP_not_accepted	op code reason

Table 3.1: General response messages.

3.2 AUTHENTICATION

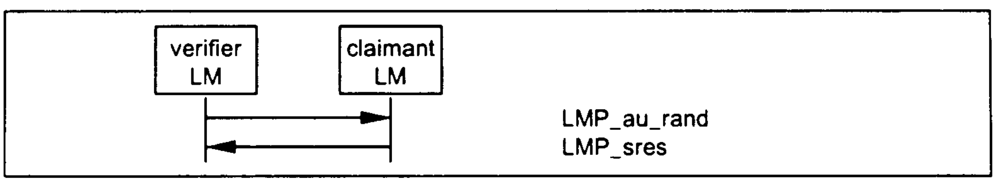
The authentication procedure is based on a challenge-response scheme as described in Baseband Specification Section 14.4, on page 169. The verifier sends an LMP_au_rand PDU which contains a random number (the challenge) to the claimant. The claimant calculates a response, which is a function of the challenge, the claimant's BD_ADDR and a secret key. The response is sent back to the verifier, which checks if the response was correct or not. How the response should be calculated is described in Baseband Specification Section 14.5.1, on page 171. A successful calculation of the authentication response requires that two devices share a secret key. How this key is created is described in Section 3.3 on page 195. Both the master and the slave can be verifiers. The following PDUs are used in the authentication procedure:

M/O	PDU	Contents
M	LMP_au_rand	random number
M	LMP_sres	authentication response

Table 3.2: PDUs used for authentication.

3.2.1 Claimant has link key

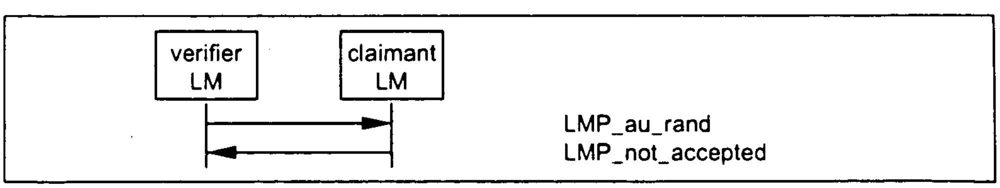
If the claimant has a link key associated with the verifier, it calculates the response and sends it to the verifier with LMP_sres. The verifier checks the response. If the response is not correct, the verifier can end the connection by sending LMP_detach with the reason code *authentication failure,* see Section 3.14 on page 207.



Sequence 1: Authentication. Claimant has link key.

3.2.2 Claimant has no link key

If the claimant does not have a link key associated with the verifier it sends LMP_not_accepted with the reason code *key missing* after receiving LMP_au_rand.



Sequence 2: Authentication fails. Claimant has no link key.

3.2.3 Repeated attempts

The scheme described in Baseband Specification Section 14.4.1, on page 170 shall be applied when an authentication fails. This will prevent an intruder from trying a large number of keys in a relatively short time.

3.3 PAIRING

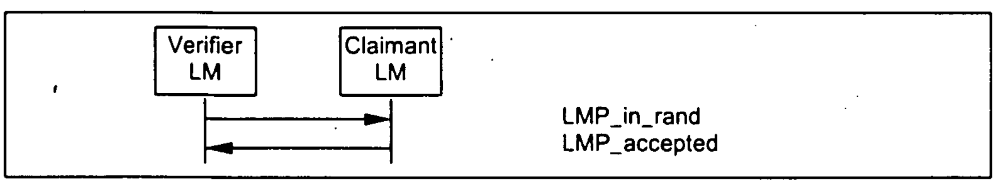
When two devices do not have a common link key an initialization key (K_{init}) is created based on a PIN and a random number. The K_{init} is created when the verifier sends LMP_in_rand to the claimant. How the K_{init} is calculated is described in Baseband Specification Section 14.5.3, on page 175. Authentication then needs to be done, whereby the calculation of the authentication response is based on K_{init} instead of the link key. After a successful authentication, the link key is created. The PDUs used in the pairing procedure are:

M/O	PDU	Contents
M	LMP_in_rand	random number
M	LMP_au_rand	random number
M	LMP_sres	authentication response
M	LMP_comb_key	random number
M	LMP_unit_key	key

Table 3.3: PDUs used for pairing.

3.3.1 Claimant accepts pairing

The verifier sends LMP_in_rand and the claimant replies with LMP_accepted. Both devices calculate K_{init}, and an authentication (see Sequence 1) based on this key needs to be done. The verifier checks the authentication response and if correct, the link key is created; see Section 3.3.4 on page 196. If the authentication response is not correct the verifier can end the connection by sending LMP_detach with the reason code *authentication failure.*

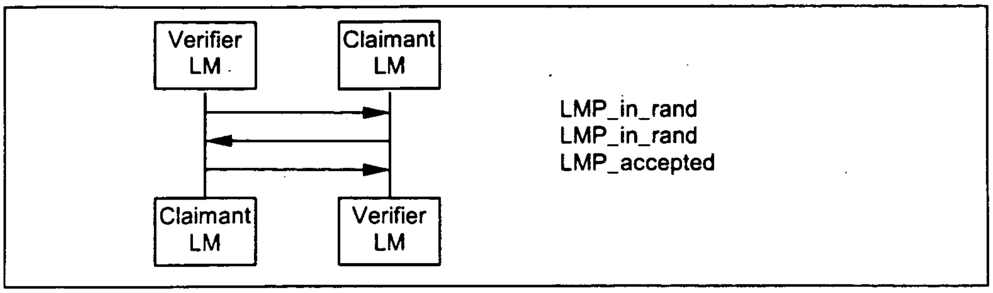


Sequence 3: Claimant accepts pairing.

3.3.2 Claimant requests to become verifier

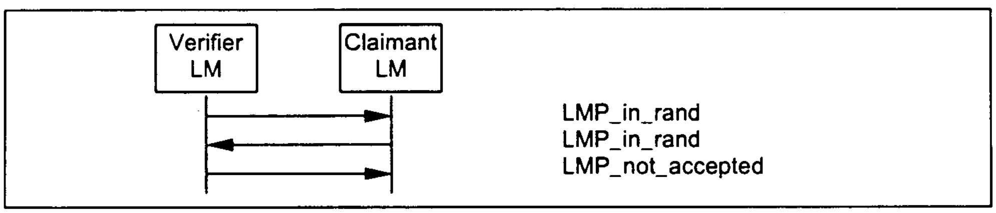
If the claimant has a fixed PIN it may request a switch of the claimant-verifier role in the pairing procedure by generating a new random number and send it

back in LMP_in_rand. If the device that started the pairing procedure has a variable PIN it must accept this and respond with LMP_accepted. The roles are then successfully switched and the pairing procedure continues as described in Section 3.3.1 on page 195.



Sequence 4: Claimant accepts pairing but requests to be verifier.

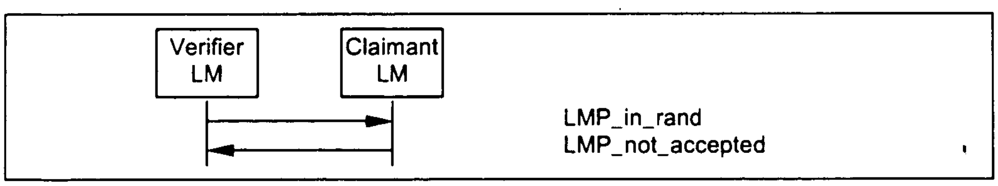
If the device that started the pairing procedure has a fixed PIN and the other device requests a role switch, the switch is rejected by sending LMP_not_accepted with the reason *pairing not allowed;* the pairing procedure is then ended.



Sequence 5: Unsuccessful switch of claimant-verifier role.

3.3.3 Claimant rejects pairing

If the claimant rejects pairing, it sends LMP_not_accepted with the reason *pairing not allowed* after receiving LMP_in_rand.



Sequence 6: Claimant rejects pairing.

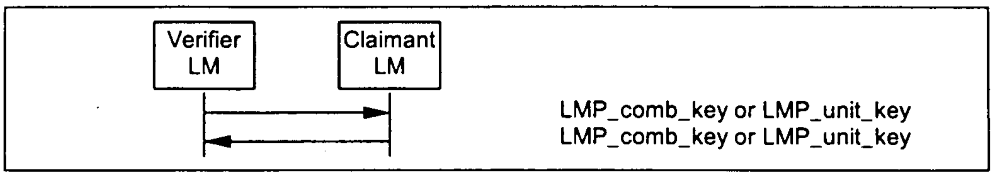
3.3.4 Creation of the link key

When the authentication is finished the link key must be created. This link key will be used in the authentication between the two units for all subsequent connections until it is changed; see Section 3.4 and Section 3.5. The link key cre-

ated in the pairing procedure will either be a combination key or one of the unit's unit keys. The following rules apply to the selection of the link key:

- if one unit sends LMP_unit_key and the other unit sends LMP_comb_key, the unit key will be the link key,
- if both units send LMP_unit_key, the master's unit key will be the link key,
- if both units send LMP_comb_key, the link key is calculated as described in Baseband Specification Section 14.2.2, on page 153.

The content of LMP_unit_key is the unit key bitwise XORed with K_{init}. The content of LMP_comb_key is LK_RAND bitwise XORed with K_{init}. Any device configured to use a combination key will store the link key in non-volatile memory.



Sequence 7: Creation of the link key.

3.3.5 Repeated attempts

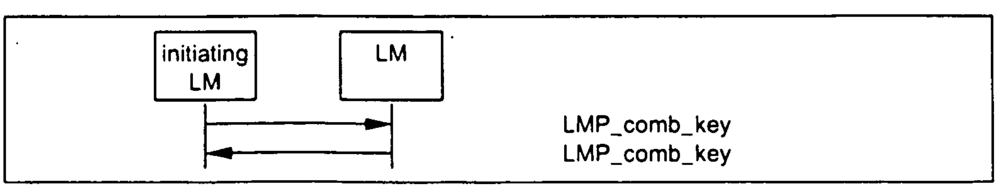
When the authentication during pairing fails because of a wrong authentication response, the same scheme is applied as in Section 3.2.3 on page 195. This prevents an intruder from trying a large number of different PINs in a relatively short time.

3.4 CHANGE LINK KEY

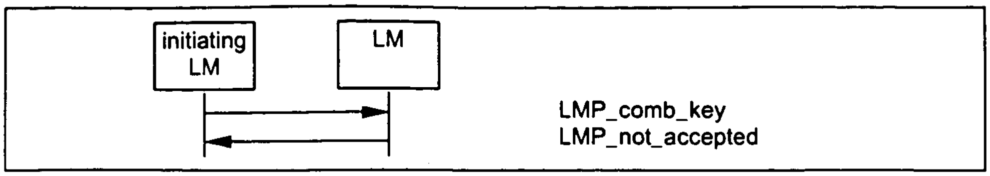
If two devices are paired and the link key is derived from combination keys, the link key can be changed. If the link key is a unit key, the units must go through the pairing procedure in order to change the link key. The contents of the PDU is protected by a bitwise XOR with the current link key.

M/O	PDU	Contents
M	LMP_comb_key	random number
M	LMP_unit_key	key

Table 3.4: PDUs used for change of link key.



Sequence 8: Successful change of the link key.



Sequence 9: Change of the link key not possible since the other unit uses a unit key.

If the change of link key is successful the new link key is stored in non-volatile memory, and the old link key is discarded. The new link key will be used as link key for all the following connections between the two devices until the link key is changed again. The new link key also becomes the current link key. It will remain the current link key until the link key is changed again, or until a temporary link key is created, see Section 3.5 on page 198.

If encryption is used on the link and the current link key is a temporary link key, the procedure of changing link key must be immediately followed by a stop of the encryption by invoking the procedure in Section 3.6.4 on page 202. Encryption can then be started again. This is to assure that encryption with encryption parameters known by other devices in the piconet is not used when the semi-permanent link key is the current link key.

3.5 CHANGE THE CURRENT LINK KEY

The current link key can be a semi-permanent link key or a temporary link key key. It can be changed temporarily, but the change is only valid for the session, see Baseband Specification Section 14.2.1, on page 151. Changing to a temporary link key is necessary if the piconet is to support encrypted broadcast.

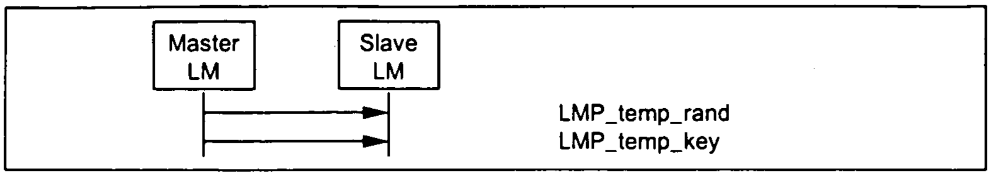
M/O	PDU	Contents
M	LMP_temp_rand	random number
M	LMP_temp_key	key
M	LMP_use_semi_permanent_key	-

Table 3.5: PDUs used to change the current link key.

3.5.1 Change to a temporary link key

In the following, we use the same terms as in Baseband Specification Section 14.2.2.8, on page 158. The master starts by creating the master key K_{master} as described in Baseband Specification (EQ 24), on page 158. Then the master issues a random number RAND and sends it to the slave in LMP_temp_rand. Both sides can then calculate an overlay denoted OVL as OVL= E_{22}(current link key, RAND, 16). Then the master sends K_{master} protected by a modulo-2 addition with OVL to the slave in LMP_temp_key. The slave, who knows OVL,

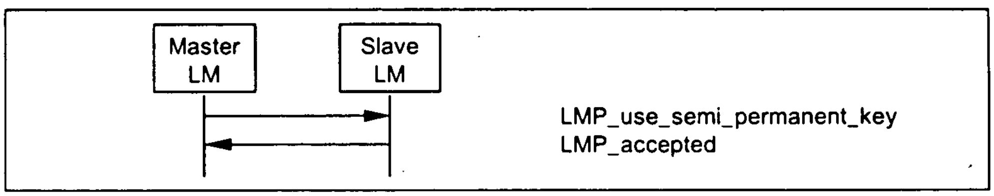
calculates K_{master}. After this, $K_{mast\ r}$ becomes the current link key. It will be the current link key until a new temporary key is created or until the link key is changed, see Section 3.4 on page 197.



Sequence 10: Change to a temporary link key.

3.5.2 Make the semi-permanent link key the current link key

After the current link key has been changed to K_{master}, this change can be undone and the semi-permanent link key becomes the current link key again. If encryption is used on the link, the procedure of going back to the semi-permanent link key must be immediately followed by a stop of the encryption by invoking the procedure described in Section 3.6.4 on page 202. Encryption can then be started again. This is to assure that encryption with encryption parameters known by other devices in the piconet is not used when the semi-permanent link key is the current link key.



Sequence 11: Link key changed to the semi-permanent link key.

3.6 ENCRYPTION

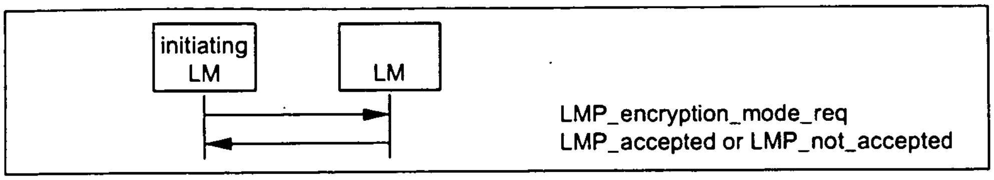
If at least one authentication has been performed encryption may be used. If the master wants all slaves in the piconet to use the same encryption parameters it must issue a temporary key (K_{master}) and make this key the current link key for all slaves in the piconet before encryption is started, see Section 3.5 on page 198. This is necessary if broadcast packets should be encrypted.

M/O	PDU	Contents
O	LMP_encryption_mode_req	encryption mode
O	LMP_encryption_key_size_req	key size
O	LMP_start_encryption_req	random number
O	LMP_stop_encryption_req	-

Table 3.6: PDUs used for handling encryption.

3.6.1 Encryption mode

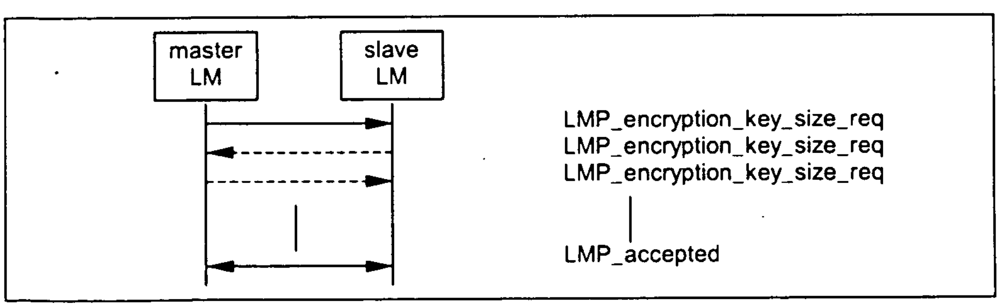
First of all the master and the slave must agree upon whether to use encryption or not and if encryption shall only apply to point-to-point packets or if encryption shall apply to both point-to-point packets and broadcast packets. If master and slave agree on the encryption mode, the master continues to give more detailed information about the encryption.



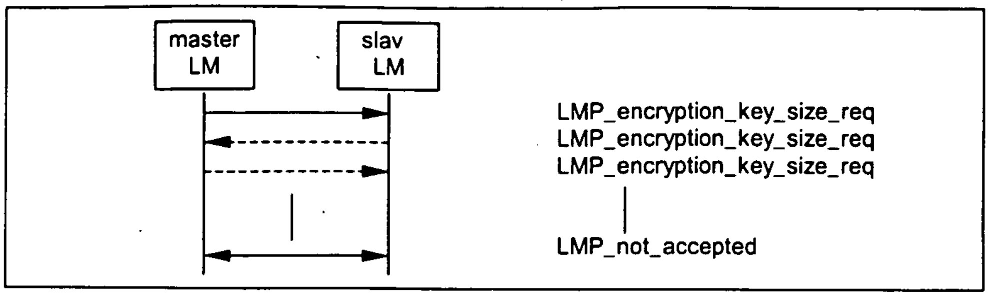
Sequence 12: Negotiation for encryption mode.

3.6.2 Encryption key size

The next step is to determine the size of the encryption key. In the following we use the same terms as in Baseband Specification Section 14.3.1, on page 160. The master sends LMP_encryption_key_size_req including the suggested key size Lsug, m, which is initially equal to $L_{max,\ m}$. If $L_{min,\ s} \leq L_{sug,\ m}$ and the slave supports $L_{sug,\ m}$ it responds with LMP_accepted and $L_{sug,\ m}$ will be used as the key size. If both conditions are not fulfilled the slave sends back LMP_encryption_key_size_req including the slave's suggested key size $L_{sug,\ s}$. This value is the slave's largest supported key size that is less than $L_{sug,\ m}$. Then the master performs the corresponding test on the slave's suggestion. This procedure is repeated until a key size agreement is reached or it becomes clear that no such agreement can be reached. If an agreement is reached a unit sends LMP_accepted and the key size in the last LMP_encryption_key_size_req will be used. After this, the encryption is started; see Section 3.6.3 on page 201. If an agreement is not reached a unit sends LMP_not_accepted with the reason code *Unsupported parameter value* and the units are not allowed to communicate using Bluetooth link encryption.".



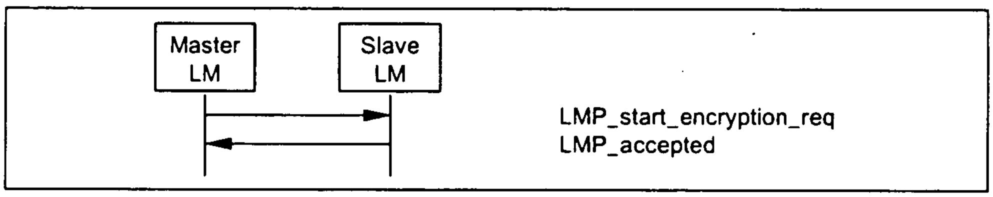
Sequence 13: Encryption key size negotiation successful.



Sequence 14: Encryption key size negotiation failed.

3.6.3 Start encryption

Finally, encryption is started. The master issues the random number EN_RAND and calculates the encryption key as $K_c=E_3$(current link key, EN_RAND, COF). See Baseband Specification Section 14.2.2.5, on page 156 and 14.2.2.2 for the definition of the COF. The random number must be the same for all slaves if the piconet should support encrypted broadcast. Then the master sends LMP_start_encryption_req, which includes EN_RAND. The slave calculates K_c when this message is received and acknowledges with LMP_accepted. On both sides, K_c and EN_RAND are used as input to the encryption algorithm E_0.



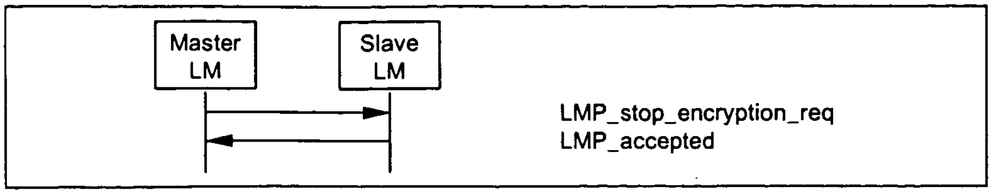
Sequence 15: Start of encryption.

Before starting encryption, higher-layer data traffic must be temporarily stopped to prevent reception of corrupt data. The start of encryption will be done in three steps:

1. Master is configured to transmit unencrypted packets, but to receive encrypted packets.
2. Slave is configured to transmit and receive encrypted packets.
3. Master is configured to transmit and receive encrypted packets.

Between step 1 and step 2, master-to-slave transmission is possible. This is when LMP_start_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3, slave-to-master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

3.6.4 Stop encryption



Sequence 16: Stop of encryption.

Before stopping encryption, higher-layer data traffic must be temporarily stopped to prevent reception of corrupt data. Stopping of encryption is then done in three steps, similar to the procedure for starting encryption.

1. Master is configured to transmit encrypted packets, but to receive unencrypted packets.
2. Slave is configured to transmit and receive unencrypted packets.
3. Master is configured to transmit and receive unencrypted packets.

Between step 1 and step 2 master to slave transmission is possible. This is when LMP_stop_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3 slave to master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

3.6.5 Change encryption mode, key or random number

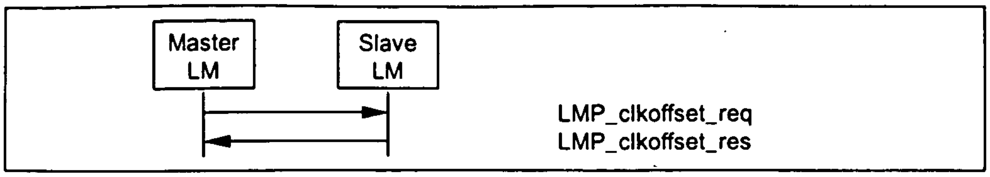
If the encryption mode, encryption key or encryption random number need to be changed, encryption must first be stopped and then re-started with the new parameters.

3.7 CLOCK OFFSET REQUEST

When a slave receives the FHS packet, the difference is computed between its own clock and the master's clock included in the payload of the FHS packet. The clock offset is also updated each time a packet is received from the master. The master can request this clock offset anytime during the connection. By saving this clock offset the master knows on what RF channel the slave wakes up to PAGE SCAN after it has left the piconet. This can be used to speed up the paging time the next time the same device is paged.

M/O	PDU	Contents
M	LMP_clkoffset_req	-
M	LMP_clkoffset_res	clock offset

Table 3.7: PDUs used for clock offset request.



Sequence 17: Clock offset requested.

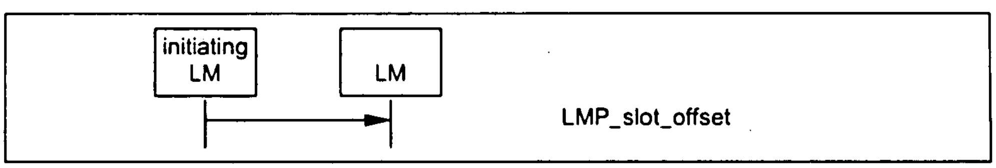
3.8 SLOT OFFSET INFORMATION

With LMP_slot_offset the information about the difference between the slot boundaries in different piconets is transmitted. This PDU carries the parameters slot offset and BD_ADDR. The slot offset is the time in µs between the start of the master's TX slot in the piconet where the PDU is transmitted and the start of the master's TX slot in the piconet where the BD_ADDR device is master.

Before doing a master-slave switch, see Section 3.12 on page 206, this PDU shall be transmitted from the device that becomes master in the switch procedure. If the master initiates the switch procedure, the slave sends LMP_slot_offset before sending LMP_accepted. If the slave initiates the switch procedure, the slave sends LMP_slot_offset before sending LMP_switch_req. The PDU can also be useful in inter-piconet communications.

M/O	PDU	Contents
O	LMP_slot_offset	slot offset BD_ADDR

Table 3.8: PDU used for slot offset information.



Sequence 18: Slot offset information is sent.

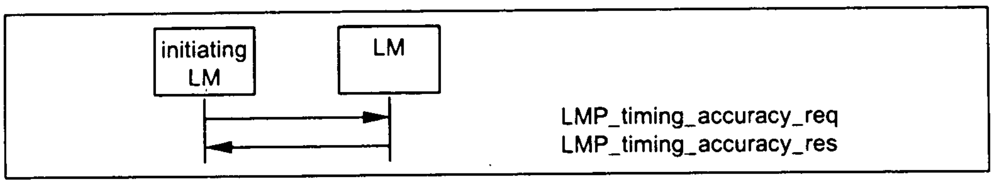
3.9 TIMING ACCURACY INFORMATION REQUEST

LMP supports requests for the timing accuracy. This information can be used to minimize the scan window for a given hold time when returning from hold and to extend the maximum hold time. It can also be used to minimize the scan window when scanning for the sniff mode slots or the park mode beacon packets. The timing accuracy parameters returned are the long term drift measured in ppm and the long term jitter measured in µs of the clock used during hold, sniff and park mode. These parameters are fixed for a certain device and must be identical when requested several times. If a device does not support the tim-

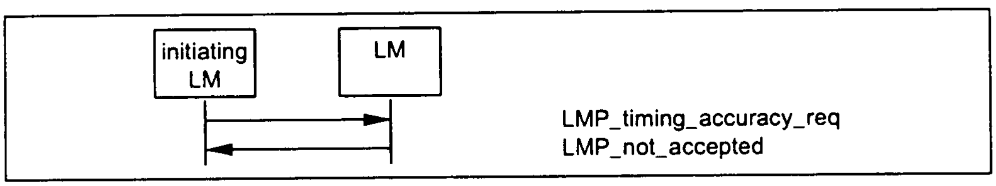
ing accuracy information it sends LMP_not_accepted with the reason code *unsupported LMP feature* when the request is received. The requesting device must in this case assume worst case values (drift=250ppm and jitter=10µs).

M/O	PDU	Contents
O	LMP_timing_accuracy_req	-
O	LMP_timing_accuracy_res	drift jitter

Table 3.9: PDUs used for requesting timing accuracy information.



Sequence 19: The requested device supports timing accuracy information.



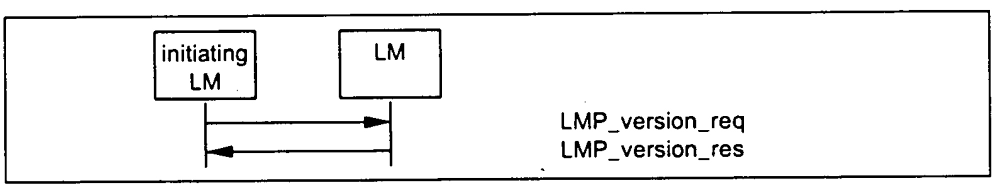
Sequence 20: The requested device does not support timing accuracy information.

3.10 LMP VERSION

LMP supports requests for the version of the LM protocol. The requested device will send a response with three parameters: VersNr, CompId and SubVersNr. VersNr specifies the version of the Bluetooth LMP specification that the device supports. CompId is used to track possible problems with the lower Bluetooth layers. All companies that create a unique implementation of the Link Manager shall have their own CompId. The same company is also responsible for the administration and maintenance of the SubVersNr. It is recommended that each company has a unique SubVersNr for each RF/BB/LM implementation. For a given VersNr and CompId, the values of the SubVersNr must increase each time a new implementation is released. For both CompId and SubVersNr the value 0xFFFF means that no valid number applies. There is no ability to negotiate the version of the LMP. The sequence below is only used to exchange the parameters.

M/O	PDU	Contents
M	LMP_version_req	VersNr CompId SubVersNr
M	LMP_version_res	VersNr CompId SubVersNr

Table 3.10: PDUs used for LMP version request.



Sequence 21: Request for LMP version.

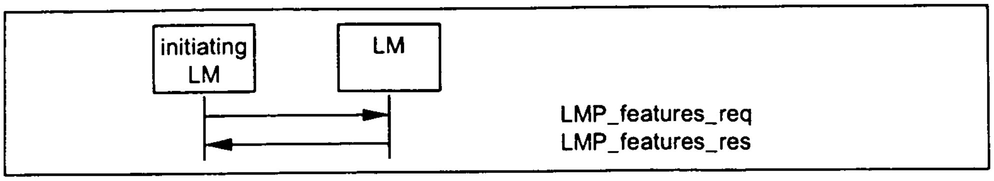
3.11 SUPPORTED FEATURES

The Bluetooth radio and link controller may support only a subset of the packet types and features described in Baseband Specification and Radio Specification. The PDU LMP_features_req and LMP_features_res are used to exchange this information. A device may not send any packets other than ID, FHS, NULL, POLL, DM1 or DH1 before it is aware of the supported features of the other device. After the features request has been carried out, the intersection of the supported packet types for both sides may also be transmitted. Whenever a request is issued, it must be compatible with the supported features of the other device. For instance, when establishing an SCO link the initiator may not propose to use HV3 packets if that packet type is not supported by the other device. Exceptions to this rule are LMP switch reg and LMP slot offset, which can be sent as the first LMP messages when two Bluetooth

devices have been connected and before the requesting side is aware of the other side's features (switch is an optional feature).

M/O	PDU	Contents
M	LMP_features_req	features
M	LMP_features_res	features

Table 3.11: PDUs used for features request.



Sequence 22: Request for supported features.

3.12 SWITCH OF MASTER-SLAVE ROLE

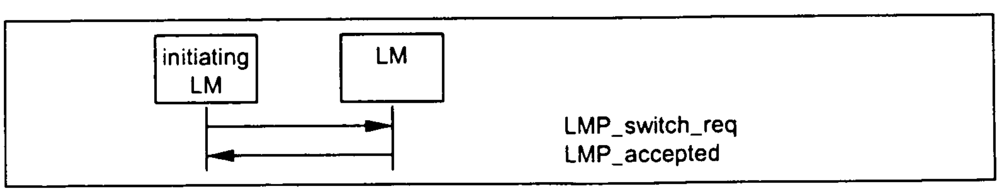
Since the paging device always becomes the master of the piconet, a switch of the master-slave role is sometimes needed, see Baseband Specification Section 10.9.3, on page 123. Suppose device A is slave and device B is master. The device that initiates the switch finalizes the transmission of the current L2CAP message and then sends LMP_switch_req.

If the switch is accepted, the other device finalizes the transmission of the current L2CAP message and then responds with LMP_accepted. After this, the procedure described from the 2nd bullet in Baseband Specification Section 10.9.3, on page 123 is carried out.

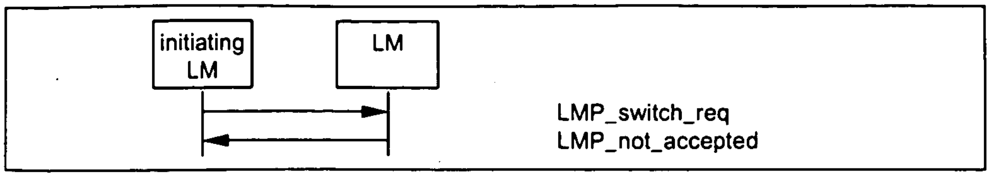
If the switch is rejected, the other device responds with LMP_not_accepted and no switch is performed.

M/O	PDU	Contents
O	LMP_switch_req	-

Table 3.12: PDU used for master slave switch.



Sequence 23: Master-slave switch accepted.



Sequence 24: Master-slave switch not accepted.

3.13 NAME REQUEST

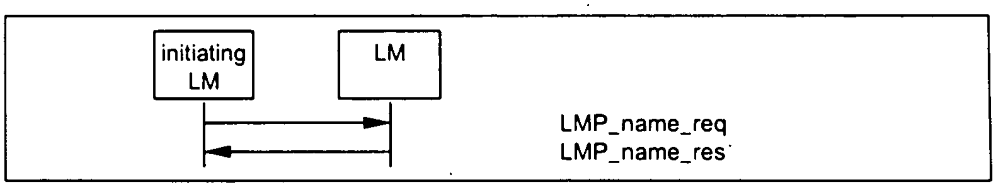
LMP supports name request to another Bluetooth device. The name is a user-friendly name associated with the Bluetooth device and consists of a maximum of 248 bytes coded according to the UTF-8 standard. The name is fragmented over one or more DM1 packets. When the LMP_name_req is sent, a name offset indicates which fragment is expected. The corresponding LMP_name_res carries the same name offset, the name length indicating the total number of bytes in the name of the Bluetooth device and the name fragment, where:

- name fragment(N) = name(N + name offset), if (N + name offset) < name length
- name fragment(N) = 0 ,otherwise.

Here $0 \leq N \leq 13$. In the first sent LMP_name_req, name offset=0. Sequence 25 is then repeated until the initiator has collected all fragments of the name.

M/O	PDU	Contents
M	LMP_name_req	name offset
M	LMP_name_res	name offset name length name fragment

Table 3.13: PDUs used for name request.



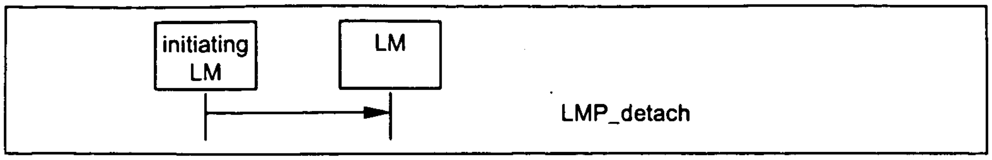
Sequence 25: Device's name requested and it responses.

3.14 DETACH

The connection between two Bluetooth devices can be closed anytime by the master or the slave. A reason parameter is included in the message to inform the other party of why the connection is closed.

M/O	PDU	Contents
M	LMP_detach	reason

Table 3.14: PDU used for detach.



Sequence 26: Connection closed by sending LMP_detach.

3.15 HOLD MODE

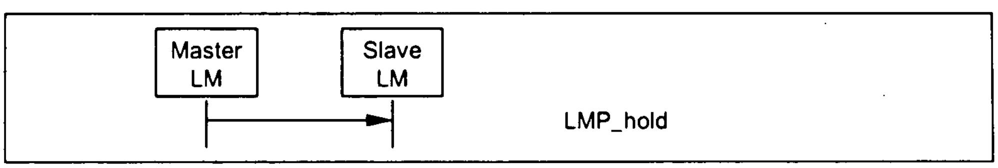
The ACL link of a connection between two Bluetooth devices can be placed in hold mode for a specified hold time. During this time no ACL packets will be transmitted from the master. The hold mode is typically entered when there is no need to send data for a relatively long time. The transceiver can then be turned off in order to save power. But the hold mode can also be used if a device wants to discover or be discovered by other Bluetooth devices, or wants to join other piconets. What a device actually does during the hold time is not controlled by the hold message, but it is up to each device to decide.

M/O	PDU	Contents
O	LMP_hold	hold time
O	LMP_hold_req	hold time

Table 3.15: PDUs used for hold mode.

3.15.1 Master forces hold mode

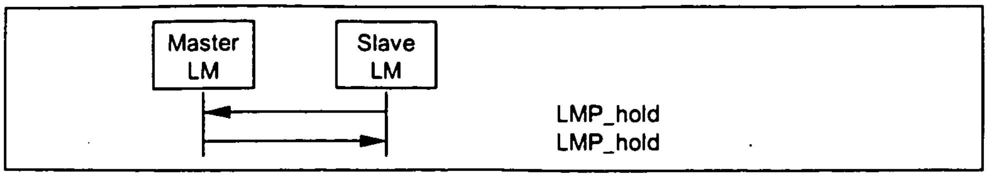
The master can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the master forces hold mode cannot be longer than any hold time the slave has previously accepted when there was a request for hold mode.



Sequence 27: Master forces slave into hold mode.

3.15.2 Slave forces hold mode

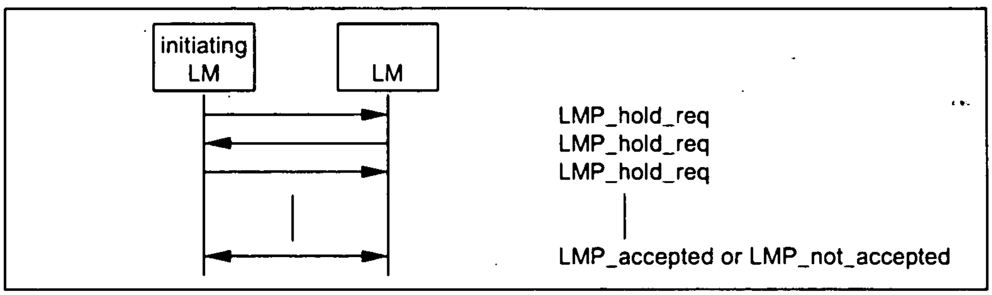
The slave can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the slave forces hold mode cannot be longer than any hold time the master has previously accepted when there was a request for hold mode.



Sequence 28: Slave forces master into hold mode.

3.15.3 Master or slave requests hold mode

The master or the slave can request to enter hold mode. Upon receipt of the request, the same request with modified parameters can be returned or the negotiation can be terminated. If an agreement is seen LMP_accepted terminates the negotiation and the ACL link is placed in hold mode. If no agreement is seen, LMP_not_accepted with the reason code *unsupported parameter value* terminates the negotiation and hold mode is not entered.



Sequence 29: Negotiation for hold mode.

3.16 SNIFF MODE

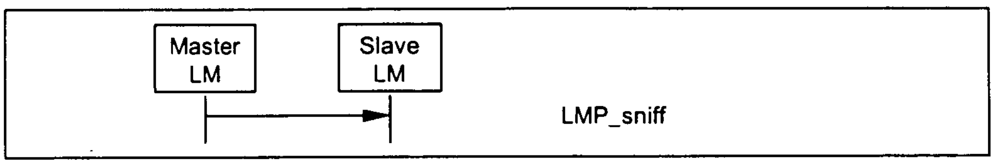
To enter sniff mode, master and slave negotiate a sniff interval T_{sniff} and a sniff offset, D_{sniff}, which specifies the timing of the sniff slots. The offset determines the time of the first sniff slot; after that the sniff slots follows periodically with the sniff interval T_{sniff}. To avoid problems with a clock wrap-around during the initialization, one of two options is chosen for the calculation of the first sniff slot. A timing control flag in the message from the master indicates this. Note: Only bit1 of this field is valid.

When the link is in sniff mode the master can only start a transmission in the sniff slot. Two parameters control the listening activity in the slave. The sniff attempt parameter determines for how many slots the slave must listen, beginning at the sniff slot, even if it does not receive a packet with its own AM address. The sniff timeout parameter determines for how many additional slots the slave must listen if it continues to receive only packets with its own AM address.

M/O	PDU	Contents
O	LMP_sniff	timing control flags D_{sniff} T_{sniff} sniff attempt sniff timeout
O	LMP_sniff_req	timing control flags D_{sniff} T_{sniff} sniff attempt sniff timeout
O	LMP_unsniff_req	-

Table 3.16: PDUs used for sniff mode.

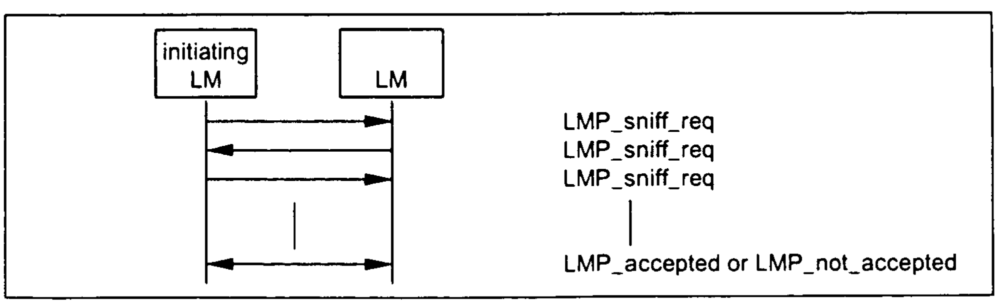
3.16.1 Master forces a slave into sniff mode



Sequence 30: Master forces slave into sniff mode.

3.16.2 Master or slave requests sniff mode

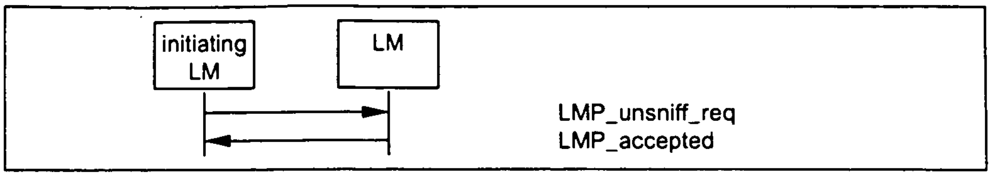
The master or the slave can request to enter sniff mode. Upon receipt of the request, the same request with modified parameters can be returned or the negotiation can be terminated. If an agreement is seen LMP_accepted terminates the negotiation and the ACL link is placed in sniff mode. If no agreement is seen, LMP_not_accepted with the reason code *unsupported parameter value* terminates the negotiation and sniff mode is not entered.



Sequence 31: Negotiation for sniff mode.

3.16.3 Moving a slave from sniff mode to active mode

Sniff mode is ended by sending the PDU LMP_unsniff_req. The requested device must reply with LMP_accepted. If the slave requests it will enter active mode after receiving LMP_accepted. If the master requests, the slave will enter active mode after receiving LMP_unsniff_req.



Sequence 32: Slave moved from sniff mode to active mode.

3.17 PARK MODE

If a slave does not need to participate in the channel, but still should be FH-synchronized, it can be placed in park mode. In this mode the device gives up its AM_ADDR but still re-synchronizes to the channel by waking up at the beacon instants separated by the beacon interval. The beacon interval, a beacon offset and a flag indicating how the first beacon instant is calculated determine the first beacon instant. After this the beacon instants follow periodically at the predetermined beacon interval. At the beacon instant the parked slave can be activated again by the master, the master can change the park mode parameters, transmit broadcast information or let the parked slaves request access to the channel.

All PDUs sent from the master to the parked slaves are broadcast. These PDUs (LMP_set_broadcast_scan_window, LMP_modify_beacon, LMP_unpark_BD_addr_req and LMP_unpark_PM_addr_req) are the only PDUs that can be sent to a slave in park mode and the only PDUs that can be broadcast. To increase reliability for broadcast, the packets are made as short as possible. Therefore the format for these LMP PDUs are somewhat different. The parameters are not always byte-aligned and the length of the PDUs is variable.

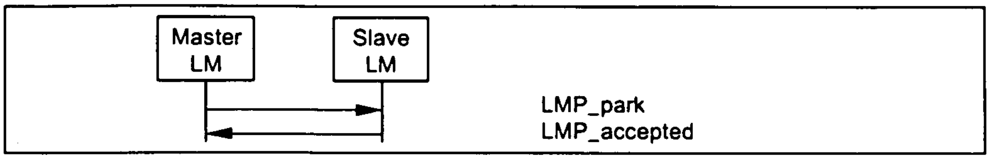
The messages for controlling the park mode include many parameters, which are all defined in Baseband Specification Section 10.8.4, on page 115. When a slave is placed in park mode it is assigned a unique PM_ADDR, which can be used by the master to unpark that slave. The all-zero PM_ADDR has a special meaning; it is not a valid PM_ADDR. If a device is assigned this PM_ADDR, it must be identified with its BD_ADDR when it is unparked by the master.

M/O	PDU	Contents
O	LMP_park_req	-
O	LMP_park	timing control flags D_B T_B N_B Δ_B PM_ADDR AR_ADDR N_{Bsleep} D_{Bsleep} D_{access} T_{access} $N_{acc\text{-}slots}$ N_{poll} M_{access} access scheme
O	LMP_set_broadcast_scan_window	timing control flags D_B (optional) broadcast scan window
O	LMP_modify_beacon	timing control flags D_B (optional) T_B N_B Δ_B D_{access} T_{access} $N_{acc\text{-}slots}$ N_{poll} M_{access} access scheme
O	LMP_unpark_PM_ADDR_req	timing control flags D_B (optional) AM_ADDR PM_ADDR AM_ADDR (optional) PM_ADDR (optional) (totally 1-7 pairs of AM_ADDR, PM_ADDR)
O	LMP_unpark_BD_ADDR_req	timing control flags D_B (optional) AM_ADDR BD_ADDR AM_ADDR (optional) BD_ADDR (optional)

Table 3.17: PDUs used for park mode.

3.17.1 Master forces a slave into park mode

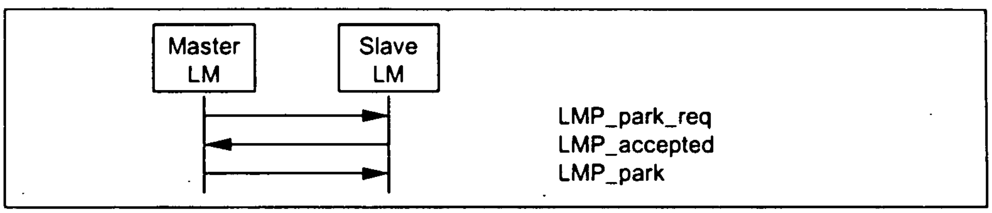
The master can force park mode. The master finalizes the transmission of the current L2CAP message and then sends LMP_park. When this PDU is received by the slave, it finalizes the transmission of the current L2CAP message and then sends LMP_accepted.



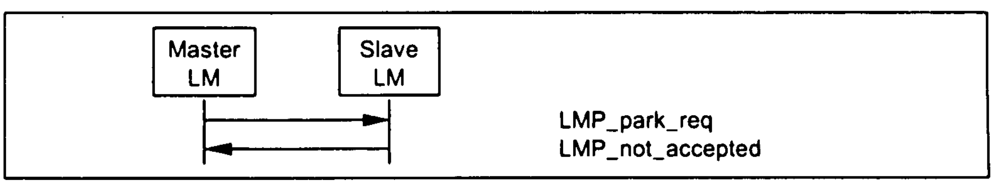
Sequence 33: Slave forced into park mode.

3.17.2 Master requests slave to enter park mode

The master can request park mode. The master finalizes the transmission of the current L2CAP message and then sends LMP_park_req. If the slave accepts to enter park mode it finalizes the transmission of the current L2CAP message and then responds with LMP_accepted. Finally the master sends LMP_park. If the slave rejects park mode it sends LMP_not_accepted.



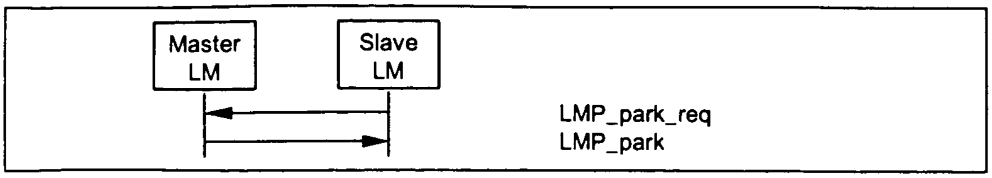
Sequence 34: Slave accepts to be placed in park mode.



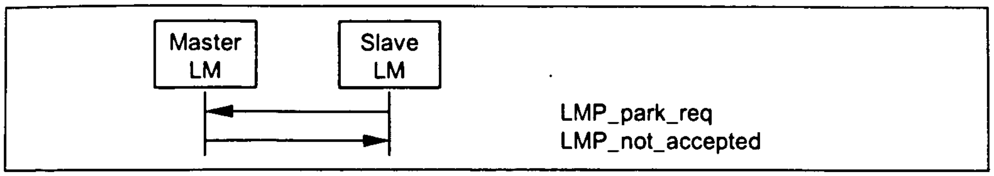
Sequence 35: Slave rejects to be placed in park mode.

3.17.3 Slave requests to be placed in park mode

The slave can request park mode. The slave finalizes the transmission of the current L2CAP message and then sends LMP_park_req. If the master accepts park mode it finalizes the transmission of the current L2CAP message and then sends LMP_park. If the master rejects park mode it sends LMP_not_accepted.



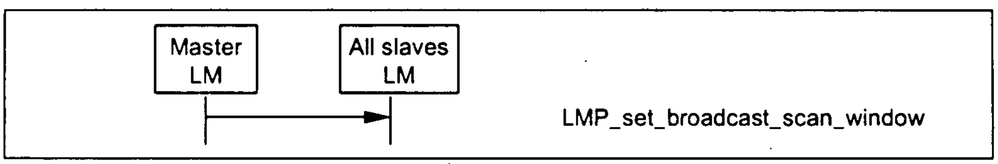
Sequence 36: Master accepts and places slave in park mode.



Sequence 37: Master rejects to place slave in park mode.

3.17.4 Master sets up broadcast scan window

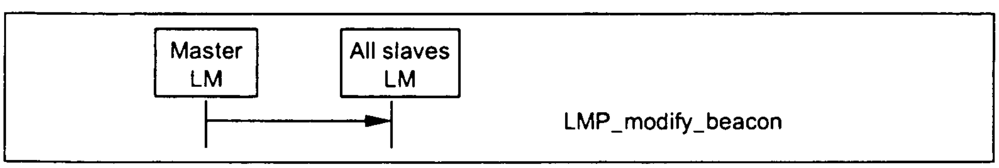
If more broadcast capacity is needed than the beacon train, the master can indicate to the slaves that more broadcast information will follow the beacon train by sending LMP_set_broadcast_scan_window. This message is always sent in a broadcast packet at the beacon slot(s). The scan window starts in the beacon instant and is only valid for the current beacon.



Sequence 38: Master notifies all slaves of increase in broadcast capacity.

3.17.5 Master modifies beacon parameters

When the beacon parameters change the master notifies the parked slaves of this by sending LMP_modify_beacon. This message is always sent in a broadcast packet at the beacon slot(s).



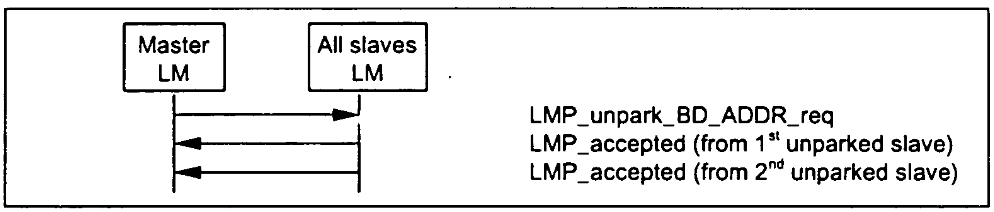
Sequence 39: Master modifies beacon parameters.

3.17.6 Unparking slaves

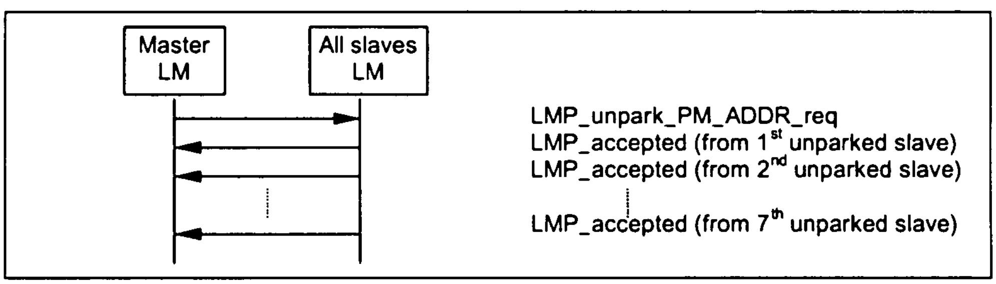
The master can unpark one or many slaves by sending a broadcast LMP message including the PM_ADDR or the BD_ADDR of the device(s) it wishes to

unpark at the beacon slot(s). This message also includes the AM_ADDR that the master assigns to the slave(s). After sending this message, the master must check the success of the unpark by polling each unparked slave, i.e. sending POLL packets, so that the slave is granted access to the channel. The unparked slave must then send a response with LMP_accepted. If this message is not received from the slave within a certain time after the master sent the unpark message, the unpark failed and the master must consider the slave as still being in park mode.

One message is used where the parked device is identified with the PM_ADDR, and another message is used where it is identified with the BD_ADDR. Both messages have variable length depending on the number of slaves the master unparks. For each slave the master wishes to unpark an AM_ADDR followed by the PM/BD_ADDR of the device that is assigned this AM_ADDR is included in the payload. If the slaves are identified with the PM_ADDR a maximum of 7 slaves can be unparked with the same message. If they are identified with the BD_ADDR a maximum of 2 slaves can be unparked with the same message.



Sequence 40: Master unparks slaves addressed with their BD_ADDR.



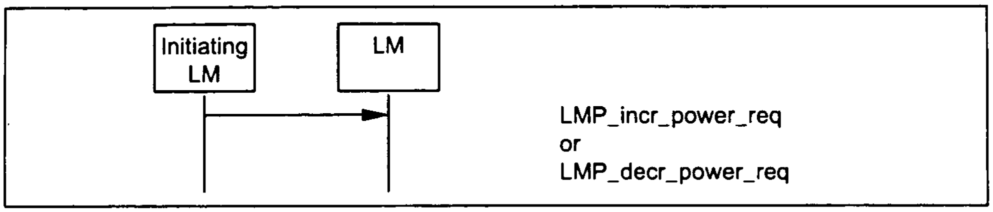
Sequence 41: Master unparks slaves addressed with their PM_ADDR.

3.18 POWER CONTROL

If the RSSI value differs too much from the preferred value of a Bluetooth device, it can request an increase or a decrease of the other device's TX power. Upon receipt of this message, the output power is increased or decreased one step. See Radio Specification Section 3.1, on page 21 for the definition of the step size. At the master side the TX power is completely independent for different slaves; a request from one slave can only effect the master's TX power for that same slave.

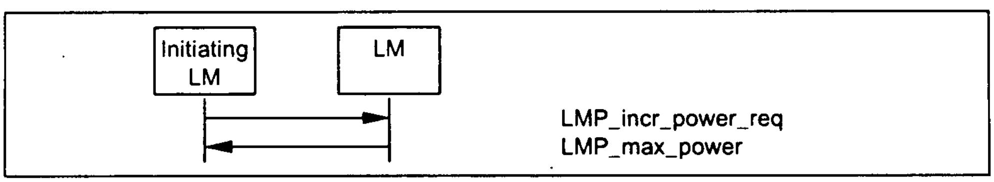
M/O	PDU	Contents
O	LMP_incr_power_req	for future use (1 Byte)
O	LMP_decr_power_req	for future use (1 Byte)
O	LMP_max_power	-
O	LMP_min_power	

Table 3.18: PDUs used for power control.

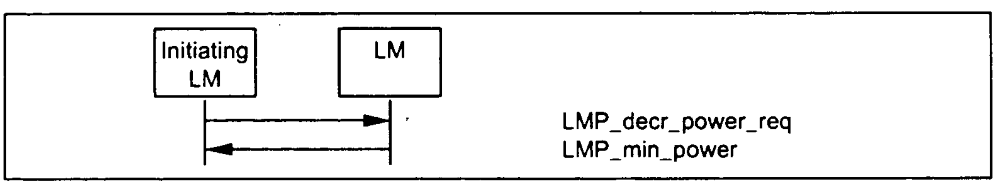


Sequence 42: A device requests a change of the other device's TX power.

If the receiver of LMP_incr_power_req already transmits at maximum power LMP_max_power is returned. The device may then only request an increase again after having requested a decrease at least once. Similarly, if the receiver of LMP_decr_power_req already transmits at minimum power then LMP_min_power is returned and the device may only request a decrease again after having requested an increase at least once.



Sequence 43: The TX power cannot be increased.



Sequence 44: The TX power cannot be decreased.

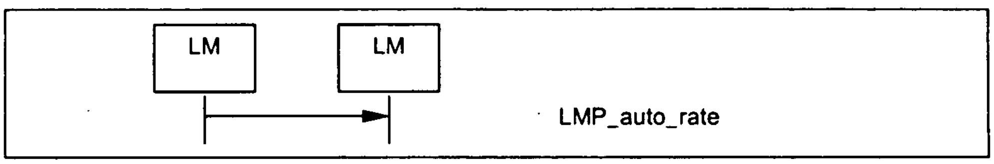
One byte is reserved in LMP_incr/decr_power_req for future use. It could, for example, be the mismatch between preferred and measured RSSI. The receiver of LMP_incr/decr_power_req could then use this value to adjust to the correct power at once, instead of only changing it one step for each request. The parameter value must be 0x00 for all versions of LMP where this parameter is not yet defined.

3.19 CHANNEL QUALITY-DRIVEN CHANGE BETWEEN DM AND DH

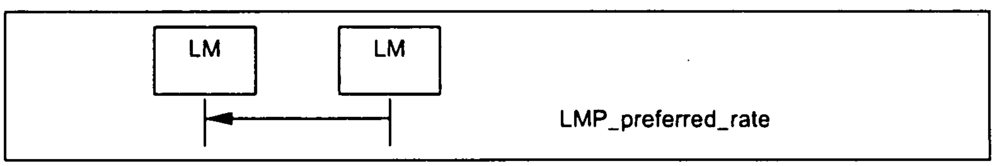
A device is configured to always use DM packets or to always use DH packets or to automatically adjust its packet type according to the quality of the channel. Nevertheless, all devices are capable of transmitting either DM or DH packets. The difference between DM and DH is that the payload in a DM packet is protected with a 2/3 FEC code, whereas the payload of a DH is not protected with any FEC. If a device wants to automatically adjust between DM and DH it sends LMP_auto_rate to the other device. Based upon quality measures in LC, the device determines if throughput will be increased by a change of packet type. If so, LMP_preferred_rate is sent to the other device. The PDUs used for this are:

M/O	PDU	Contents
O	LMP_auto_rate	-
O	LMP_preferred_rate	data rate

Table 3.19: PDUs used for quality driven change of the data rate.



Sequence 45: The left-hand unit is configured to automatically change between DM and DH.



Sequence 46: The right-hand device orders the left-hand device to change data rate.

3.20 QUALITY OF SERVICE (Q S)

The Link Manager provides Quality of Service capabilities. A poll interval, which is defined as the maximum time between subsequent transmissions from the master to a particular slave, is used to support bandwidth allocation and latency control. The poll interval is guaranteed except when there are collisions with page, page scan, inquiry and inquiry scan.

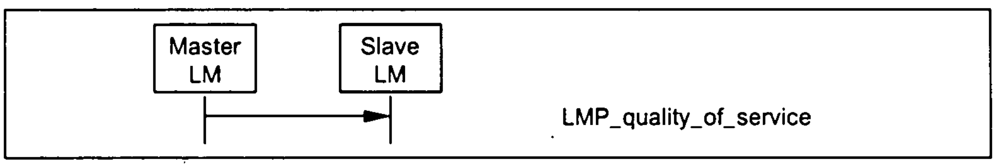
In addition, master and slave negotiate the number of repetitions for broadcast packets (NBC), see Baseband Specification Section 5.3, on page 68.

M/O	PDU	Contents
M	LMP_quality_of_service	poll interval N_{BC}
M	LMP_quality_of_service_req	poll interval N_{BC}

Table 3.20: PDUs used for quality of service.

3.20.1 Master notifies slave of the quality of service

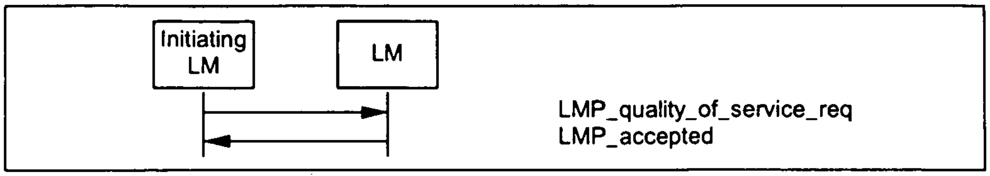
In this case the master notifies the slave of the new poll interval and N_{BC}. The slave cannot reject the notification.



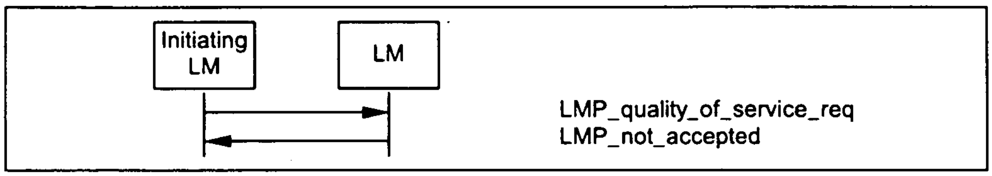
Sequence 47: Master notifies slave of new quality of service.

3.20.2 Device requests new quality of service

In this case the master or slave requests a new poll interval and N_{BC}. The parameter N_{BC} is meaningful only when it is sent by a master to a slave. For transmission of LMP_quality_of_service_req PDUs from a slave, this parameter is ignored by the master. The request can be accepted or rejected. This will allow the master and slave to dynamically negotiate the quality of service as needed.



Sequence 48: Device accepts new quality of service



Sequence 49: Device rejects new quality of service.

3.21 SCO LINKS

When a connection has been established between two Bluetooth devices the connection consists of an ACL link. One or more SCO links can then be established. The SCO link reserves slots separated by the SCO interval, T_{sco}. The first slot reserved for the SCO link is defined by T_{sco} and the SCO delay, D_{sco}. After that the SCO slots follows periodically with the SCO interval. To avoid problems with a wrap-around of the clock during initialization of the SCO link, a flag indicating how the first SCO slot should be calculated is included in a message from the master. Note: Only bit0 and bit1 of this field is valid. Each SCO link is distinguished from all other SCO links by an SCO handle. The SCO handle zero is never used.

M/O	PDU	Contents
O	LMP_SCO_link_req	SCO handle timing control flags D_{sco} T_{sco} SCO packet air mode
O	LMP_remove_SCO_link_req	SCO handle reason

Table 3.21: PDUs used for managing the SCO links.

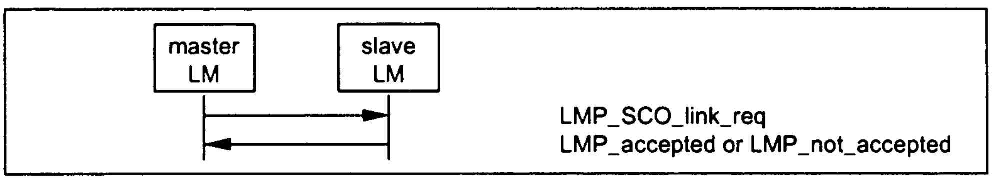
3.21.1 Master initiates an SCO link

When establishing an SCO link the master sends a request with parameters that specify the timing, packet type and coding that will be used on the SCO link. For each of the SCO packets Bluetooth supports three different voice coding formats on the air-interface: μ-law log PCM, A-law log PCM and CVSD.

The slots used for the SCO links are determined by three parameters controlled by the master: T_{sco}, D_{sco} and a flag indicating how the first SCO slot should be calculated. After the first slot, the SCO slots follows periodically with the T_{sco}.

If the slave does not accept the SCO link, but is willing to consider another possible set of SCO parameters, it can indicate what it does not accept in the error reason field of LMP_not_accepted. The master then has the possibility to issue a new request with modified parameters.

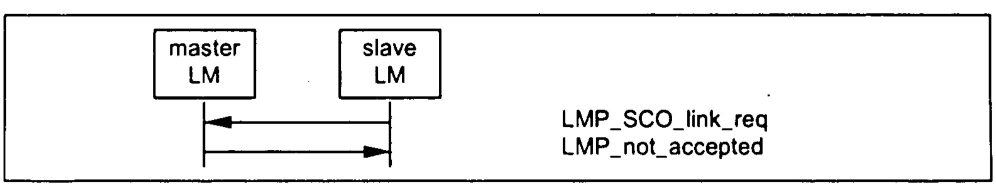
The SCO handle in the message must be different from any already existing SCO link(s).



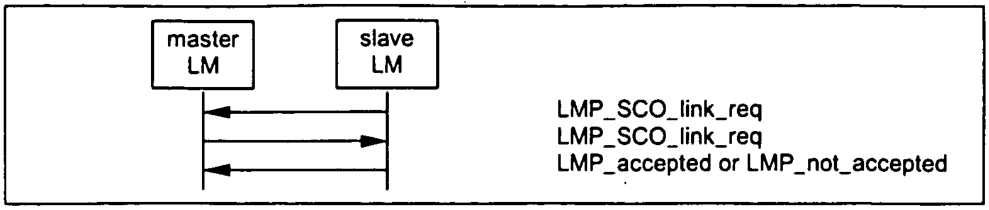
Sequence 50: Master requests an SCO link.

3.21.2 Slave initiates an SCO link

The slave can also initiate the establishment of an SCO link. The slave sends LMP_SCO_link_req, but the parameters timing control flags and D_{sco} are invalid as well as the SCO handle, which must be zero. If the master is not capable of establishing an SCO link, it replies with LMP_not_accepted. Otherwise it sends back LMP_SCO_link_req. This message includes the assigned SCO handle, D_{sco} and the timing control flags. For the other parameters, the master should try to use the same parameters as in the slave request; if the master cannot meet that request, it is allowed to use other values. The slave must then reply with LMP_accepted or LMP_not_accepted.



Sequence 51: Master rejects slave's request for an SCO link.



Sequence 52: Master accepts slave's request for an SCO link.

3.21.3 Master requests change of SCO parameters

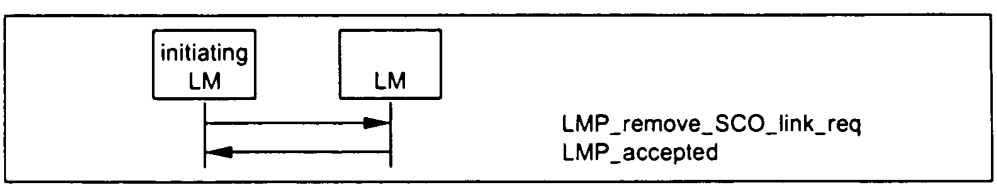
The master sends LMP_SCO_link_req, where the SCO handle is the handle of the SCO link the master wishes to change parameters for. If the slave accepts the new parameters, it replies with LMP_accepted and the SCO link will change to the new parameters. If the slave does not accept the new parameters, it replies with LMP_not_accepted and the SCO link is left unchanged. When the slave replies with LMP_not_accepted it shall indicate in the error reason parameter what it does not accept. The master can then try to change the SCO link again with modified parameters. The sequence is the same as in Section 3.21.1 on page 220.

3.21.4 Slave requests change of SCO parameters

The slave sends LMP_SCO_link_req, where the SCO handle is the handle of the SCO link the slave wishes to change parameters for. The parameters timing control flags and D_{SCO} are not valid in this message. If the master does not accept the new parameters it replies with LMP_not_accepted and the SCO link is left unchanged. If the master accepts the new parameters it replies with LMP_SCO_link_req, where it must use the same parameters as in the slave request. When receiving this message the slave replies with LMP_not_accepted if it does not accept the new parameters. The SCO link is then left unchanged. If the slave accepts the new parameters it replies with LMP_accepted and the SCO link will change to the new parameters. The sequence is the same as in Section 3.21.2 on page 220.

3.21.5 Remove an SCO link

Master or slave can remove the SCO link by sending a request including the SCO handle of the SCO link to be removed and a reason indicating why the SCO link is removed. The receiving party must respond with LMP_accepted.



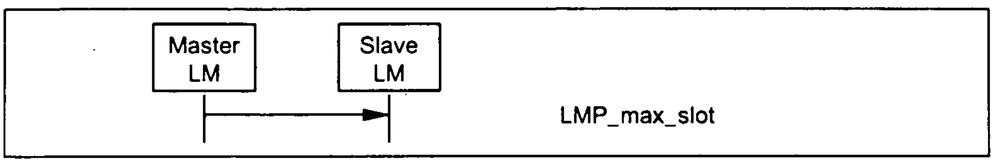
Sequence 53: SCO link removed.

3.22 CONTROL OF MULTI-SLOT PACKETS

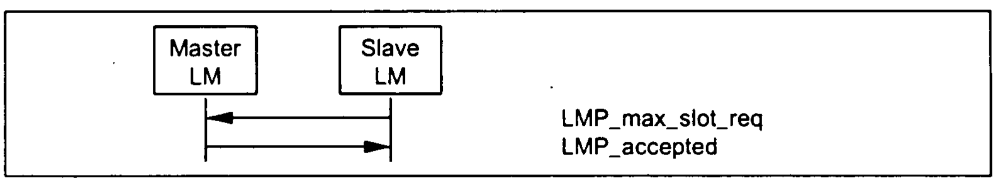
The number of slots used by a slave in its return packet can be limited. The master allows the slave to use a maximal number of slots by sending the PDU LMP_max_slots providing max slots as parameter. Each slave can request to use a maximal number of slots by sending the PDU LMP_max_slot_req providing max slots as parameter. The default value is 1 slot, i.e. if the slave has not been informed about the number of slots, it may only use 1-slot packets. Two PDUs are used for the control of multi-slot packets.

M/O	PDU	Contents
M	LMP_max_slot	max slots
M	LMP_max_slot_req	max slots

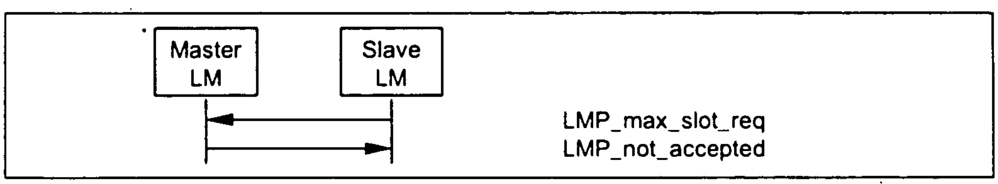
Table 3.22: PDUs used to control the use of multi-slot packets.



Sequence 54: Master allows slave to use a maximal number of slots.



Sequence 55: Slave requests to use a maximal number of slots. Master accepts.



Sequence 56: Slave requests to use a maximal number of slots. Master rejects.

3.23 PAGING SCHEME

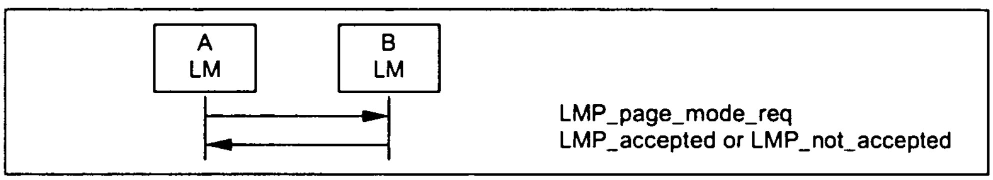
In addition to the mandatory paging scheme, Bluetooth defines optional paging schemes; see "Appendix VII" on page 999. LMP provides a means to negotiate the paging scheme, which is to be used the next time a unit is paged.

M/O	PDU	Contents
O	LMP_page_mode_req	paging scheme paging scheme settings
O	LMP_page_scan_mode_req	paging scheme paging scheme settings

Table 3.23: PDUs used to request paging scheme.

3.23.1 Page mode

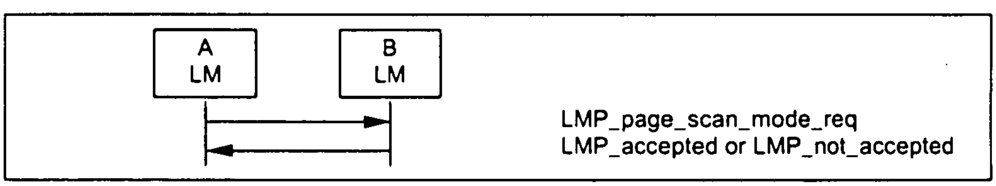
This procedure is initiated from device A and negotiates the paging scheme used when device A pages device B. Device A proposes a paging scheme including the parameters for this scheme and device B can accept or reject. On rejection the old setting is not changed. A request to switch back to the mandatory scheme may be rejected.



Sequence 57: Negotiation for page mode.

3.23.2 Page scan mode

This procedure is initiated from device A and negotiates the paging scheme used when device B pages device A. Device A proposes a paging scheme including the parameters for this scheme and device B can accept or reject. On rejection the old setting is not changed. A request to switch to the mandatory scheme must be accepted.



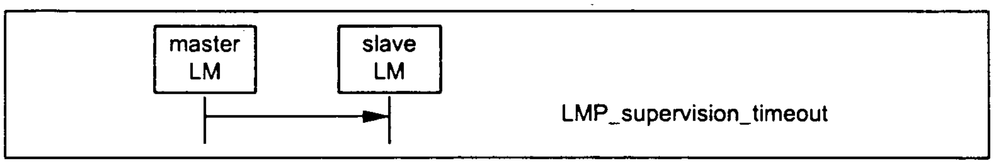
Sequence 58: Negotiation for page scan mode

3.24 LINK SUPERVISION

Each Bluetooth link has a timer that is used for link supervision. This timer is used to detect link loss caused by devices moving out of range, a device's power-down, or other similar failure cases. The scheme for link supervision is described in Baseband Specification Section 10.11, on page 126. An LMP procedure is used to set the value of the supervision timeout.

M/O	PDU	Contents
M	LMP_supervision_timeout	supervision timeout

Table 3.24: PDU used to set the supervision timeout.



Sequence 59: Setting the link supervision timeout.

4 CONNECTION ESTABLISHMENT

After the paging procedure, the master must poll the slave by sending POLL or NULL packets, with a max poll interval as defined in Table 5.5 on page 236. LMP procedures that do not require any interactions between the LM and the host at the paged unit's side can then be carried out.

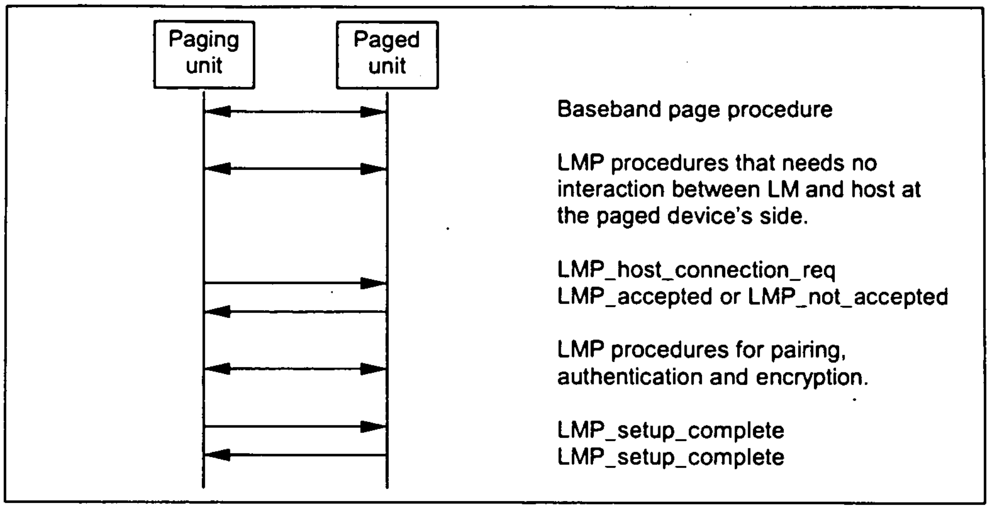


Figure 4.1: Connection establishment.

When the paging device wishes to create a connection involving layers above LM, it sends LMP_host_connection_req. When the other side receives this message, the host is informed about the incoming connection. The remote device can accept or reject the connection request by sending LMP_accepted or LMP_not_accepted.

When a device does not require any further link set-up procedures, it will send LMP_setup_complete. The device will still respond to requests from the other device. When the other device is also ready with link set-up, it will send LMP_setup_complete. After this, the first packet on a logical channel different from LMP can then be transmitted.

M/O	PDU	Contents
M	LMP_host_connection_req	-
M	LMP_setup_complete	-

Table 4.1: PDUs used for connection establishment.

5 SUMMARY OF PDUs

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_accepted	2	3	DM1/ DV	m ↔ s	op code	2
LMP_au_rand	17	11	DM1	m ↔ s	random number	2-17
LMP_auto_rate	1	35	DM1/ DV	m ↔ s	-	
LMP_clkoffset_req	1	5	DM1/ DV	m → s	-	
LMP_clkoffset_res	3	6	DM1/ DV	m ← s	clock offset	2–3
LMP_comb_key	17	9	DM1	m ↔ s	random number	2-17
LMP_decr_power_req	2	32	DM1/ DV	m ↔ s	for future use	2
LMP_detach	2	7	DM1/ DV	m ↔ s	reason	2
LMP_encryption_key_size _req	2	16	DM1/ DV	m ↔ s	key size	2
LMP_encryption_mode_ req	2	15	DM1/ DV	m ↔ s	encryption mode	2
LMP_features_req	9	39	DM1/ DV	m ↔ s	features	2-9
LMP_features_res	9	40	DM1/ DV	m ↔ s	features	2-9
LMP_host_connection_req	1	51	DM1/ DV	m ↔ s	-	
LMP_hold	3	20	DM1/ DV	m ↔ s	hold time	2-3
LMP_hold_req	3	21	DM1/ DV	m ↔ s	hold time	2-3
LMP_incr_power_req	2	31	DM1/ DV	m ↔ s	for future use	2
LMP_in_rand	17	8	DM1	m ↔ s	random number	2-17
LMP_max_power	1	33	DM1/ DV	m ↔ s	-	

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possibl direction	Contents	Position in payload
LMP_max_slot	2	45	DM1/ DV	m → s	max slots	2
LMP_max_slot_req	2	46	DM1/ DV	m ← s	max slots	2
LMP_min_power	1	34	DM1/ DV	m ↔ s	-	
LMP_modify_beacon	11 or 13	28	DM1	m → s	timing control flags	2
					D_B	3-4
					T_B	5-6
					N_B	7
					Δ_B	8
					D_{access}	9
					T_{access}	10
					$N_{acc\text{-}slots}$	11
					N_{poll}	12
					M_{access}	13:0-3
					access scheme	13:4-7
LMP_name_req	2	1	DM1/ DV	m ↔ s	name offset	2
LMP_name_res	17	2	DM1	m ↔ s	name offset	2
					name length	3
					name fragment	4-17
LMP_not_accepted	3	4	DM1/ DV	m ↔ s	op code	2
					reason	3
LMP_page_mode_req	3	53	DM1/ DV	m ↔ s	paging scheme	2
					paging scheme settings	3
LMP_page_scan_mode_ req	3	54	DM1/ DV	m ↔ s	paging scheme	2
					paging scheme settings	3

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_park	17	26	DM	m → s	timing control flags	2
					D_B	3-4
					T_B	5-6
					N_B	7
					Δ_B	8
					PM_ADDR	9
					AR_ADDR	10
					N_{Bsleep}	11
					D_{Bsleep}	12
					D_{access}	13
					T_{access}	14
					$N_{acc-slots}$	15
					N_{poll}	16
					M_{access}	17:0-3
					access scheme	17:4-7
LMP_park_req	1	25	DM1/ DV	m ↔ s	-	
LMP_preferred_rate	2	36	DM1/ DV	m ↔ s	data rate	2
LMP_quality_of_service	4	41	DM1/ DV	m → s	poll interval	2-3
					N_{BC}	4
LMP_quality_of_service_req	4	42	DM1/ DV	m ↔ s	poll interval	2-3
					N_{BC}	4
LMP_remove_SCO_link_req	3	44	DM1/ DV	m ↔ s	SCO handle	2
					reason	3

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_SCO_link_req	7	43	DM1/ DV	m ↔ s	SCO handle	2
					timing control flags	3
					D_{sco}	4
					T_{sco}	5
					SCO packet	6
					air mode	7
LMP_set_broadcast_ scan_window	4 or 6	27	DM1	m → s	timing control flags	2
					D_B	3-4
					broadcast scan window	5-6
LMP_setup_complete	1	49	DM1	m ↔ s	-	
LMP_slot_offset	9	52	DM1/ DV	m ↔ s	slot offset	2-3
					BD_ADDR	4-9
LMP_sniff	10	22	DM1	m → s	timing control flags	2
					D_{sniff}	3-4
					T_{sniff}	5-6
					sniff attempt	7-8
					sniff timeout	9-10
LMP_sniff_req	10	23	DM1	m ↔ s	timing control flags	2
					D_{sniff}	3-4
					T_{sniff}	5-6
					sniff attempt	7-8
					sniff timeout	9-10
LMP_sres	5	12	DM1/ DV	m ↔ s	authentication response	2-5
LMP_start_encryption_req	17	17	DM1	m → s	random number	2-17
LMP_stop_encryption_req	1	18	DM1/ DV	m → s	-	
LMP_supervision_timeout	3	55	DM1/ DV	m ↔ s	supervision timeout	2-3
LMP_switch_req	1	19	DM1/ DV	m ↔ s	-	

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_temp_rand	17	13	DM1	m → s	random number	2-17
LMP_temp_key	17	14	DM1	m → s	key	2-17
LMP_timing_accuracy_req	1	47	DM1/ DV	m ↔ s	-	
LMP_timing_accuracy_res	3	48	DM1/ DV	m ↔ s	drift	2
					jitter	3
LMP_unit_key	17	10	DM1	m ↔ s	key	2-17
LMP_unpark_BD_ADDR _req	variable	29	DM1	m → s	timing control flags	2
					D_B	3-4
					AM_ADDR 1st unpark	5:0-3
					AM_ADDR 2nd unpark	5:4-7
					BD_ADDR 1st unpark	6-11
					BD_ADDR 2nd unpark	12-17
LMP_unpark_PM_ADDR _req	variable	30	DM1	m → s	timing control flags	2
					D_B	3-4
					AM_ADDR 1st unpark	5:0-3
					AM_ADDR 2nd unpark	5:4-7
					PM_ADDR 1st unpark	6
					PM_ADDR 2nd unpark	7
LMP_unsniff_req	1	24	DM1/ DV	m ↔ s	-	
LMP_use_semi_ permanent_key	1	50	DM1/ DV	m → s	-	
LMP_version_req	6	37	DM1/ DV	m ↔ s	VersNr	2
					CompId	3-4
					SubVersNr	5-6
LMP_version_res	6	38	DM1/ DV	m ↔ s	VersNr	2
					CompId	3-4
					SubVersNr	5-6

Table 5.1: Coding of the different LM PDUs.

Note1: For LMP_set_broadcast_scan_window, LMP_modify_beacon, LMP_unpark_BD_ADDR_req and LMP_unpark_PM_ADDR_req the parameter

D_B is optional. This parameter is only present if bit0 of *timing control flags* is 0. If the parameter is not included, the position in payload for all parameters following D_B are decreased by 2.

Note2: For LMP_unpark_BD_ADDR the AM_ADDR and the BD_ADDR of the 2^{nd} unparked slave are optional. If only one slave is unparked *AM_ADDR 2^{nd} unpark* should be zero and *BD_ADDR 2^{nd} unpark* is left out.

Note3: For LMP_unpark_PM_ADDR the AM_ADDR and the PM_ADDR of the 2^{nd} – 7^{th} unparked slaves are optional. If N slaves are unparked, the fields up to and including the N^{th} unparked slave are present. If N is odd, the *AM_ADDR $(N+1)^{th}$ unpark* must be zero. The length of the message is x + 3N/2 if N is even and x + 3(N+1)/2 -1 if N is odd, where x = 2 or 4 depending on if the D_B is incluDed Or Not (See Note1).

5.1 DESCRIPTION OF PARAMETERS

Name	Length (bytes)	Type	Unit	Detailed
access scheme	1	u_int4		0: polling technique 1-15: Reserved
air mode	1	u_int8		0: µ-law log 1: A-law log 2: CVSD 3-255: Reserved
AM_ADDR	1	u_int4		
AR_ADDR	1	u_int8		
authentication response	4	multiple bytes		
BD_ADDR	6	multiple bytes		
broadcast scan window	2	u_int16	slots	
clock offset	2	u_int16	1.25ms	($CLKN_{16-2}$ slave - $CLKN_{16-2}$ master) mod 2^{15} MSbit of second byte not used.
CompId	2	u_int16		see BT Assigned Numbers Section 2.1 on page 1018
D_{access}	1	u_int8	slots	
D_B	2	u_int16	slots	

Table 5.2: Parameters in LM PDUs.

Name	Length (bytes)	Type	Unit	Detailed
D_{Bsleep}	1	u_int8	slots	
data rate	1	u_int8		0: medium rate 1: high rate 2-255: Reserved
drift	1	u_int8	ppm	
D_{sco}	1	u_int8	slots	
D_{sniff}	2	u_int16	slots	
encryption mode	1	u_int8		0: no encryption 1: point-to-point encryption 2: point-to-point and broadcast encryption 3 -255: Reserved
features	8	multiple bytes		See Table 5.3 on page 234
hold time	2	u_int16	slots	
jitter	1	u_int8	μs	
key	16	multiple bytes		
key size	1	u_int8	byte	
M_{access}	1	u_int4	slots	
max slots	1	u_int8	slots	
$N_{acc-slots}$	1	u_int8	slots	
name fragment	14	multiple bytes		UTF-8 characters.
name length	1	u_int8	bytes	
name offset	1	u_int8	bytes	
N_B	1	u_int8		
N_{BC}	1	u_int8		
N_{Bsleep}	1	u_int8	slots	
N_{poll}	1	u_int8	slots	
op code	1	u_int8		
paging scheme	1	u_int8		0: mandatory scheme 1: optional scheme 1 2-255: Reserved

Table 5.2: Parameters in LM PDUs.

Nam	Length (byte)	Type	Unit	D tailed
paging scheme settings	1	u_int8		For mandatory scheme: 0: R0 1: R1 2: R2 3-255: Reserved For optional scheme 1: 0: Reserved 1: R1 2: R2 3-255: Reserved
PM_ADDR	1	u_int8		
poll interval	2	u_int16	slots	
random number	16	multiple bytes		
reason	1	u_int8		See Table 5.4 on page 235.
SCO handle	1	u_int8		
SCO packet	1	u_int8		0: HV1 1: HV2 2: HV3 3-255: Reserved
slot offset	2	u_int16	µs	0 ≤ slot offset < 1250
sniff attempt	2	u_int16	slots	
sniff timeout	2	u_int16	slots	
SubVersNr	2	u_int16		Defined by each company
supervision time-out	2	u_int16	slots	
T_{access}	1	u_int8	slots	
T_B	2	u_int16	slots	
timing control flags	1	u_int8		bit0 = 0: no timing change bit0 = 1: timing change bit1 = 0: use initialization 1 bit1 = 1: use initialization 2 bit2 = 0: access window bit2 = 1: no access window bit3-7: Reserved

Table 5.2: Parameters in LM PDUs.

Nam	Length (bytes)	Type	Unit	D tail d
T_{sco}	1	u_int8	slots	
T_{sniff}	2	u_int16	slots	
VersNr	1	u_int8		0: Bluetooth LMP 1.0 1-255: Reserved
Δ_B	1	u_int8	slots	

Table 5.2: Parameters in LM PDUs.

5.1.1 Coding of features

This parameter is a bitmap with information about the Bluetooth radio-, baseband- and LMP features which a device supports. The bit shall be one if the feature is supported. The feature parameter bits that are not defined in Table 5.3 shall be zero.

Byte	Bit	Supported feature
0	0	3-slot packets
	1	5-slot packets
	2	encryption
	3	slot offset
	4	timing accuracy
	5	switch
	6	hold mode
	7	sniff mode
1	0	park mode
	1	RSSI
	2	channel quality driven data rate
	3	SCO link
	4	HV2 packets
	5	HV3 packets
	6	u-law log
	7	A-law log
2	0	CVSD
	1	paging scheme
	2	power control

Table 5.3: Coding of the parameter features.

5.1.2 List f rr r reasons

The following table contains the codes of the different error reasons used in LMP.

Reason	Description
0x05	Authentication Failure
0x06	Key Missing
0x0A	Max Number Of SCO Connections To A Device (The maximum number of SCO connections to a particle device has been reached. All allowed SCO connection handles to that device are used.)
0x0D	Host Rejected due to limited resources (The host at the remote side has rejected the connection because the remote host did not have enough additional resources to accept the connection.)
0x0E	Host Rejected due to security reasons (The host at the remote side has rejected the connection because the remote host determined that the local host did not meet its security criteria.)
0x0F	Host Rejected due to remote device is only a personal device (The host at the remote side has rejected the connection because the remote host is a personal device and will only accept the connection from one particle remote host.)
0x10	Host Timeout (Used at connection accept timeout, the host did not respond to an incoming connection attempt before the connection accept timer expired.)
0x13	Other End Terminated Connection: User Ended Connection
0x14	Other End Terminated Connection: Low Resources
0x15	Other End Terminated Connection: About to Power Off
0x16	Connection Terminated by Local Host
0x17	Repeated Attempts (An authentication or pairing attempt is made too soon after a previously failed authentication or pairing attempt.)
0x18	Pairing Not Allowed
0x19	Unknown LMP PDU
0x1A	Unsupported LMP Feature
0x1B	SCO Offset Rejected
0x1C	SCO Interval Rejected
0x1D	SCO Air Mode Rejected
0x1E	Invalid LMP Parameters
0x1F	Unspecified Error
0x20	Unsupported parameter value
0x21	Switch not allowed
0x23	LMP Error Transaction Collision
0x24	PDU not allowed

Table 5.4: List of error reasons.

5.2 DEFAULT VALUES

The Bluetooth device must use these values before anything else has been negotiated:

Parameter	Value
drift	250
jitter	10
max slots	1
poll interval	40

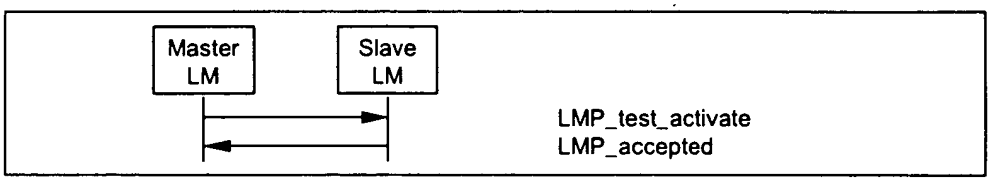
Table 5.5: Default values.

6 TEST MODES

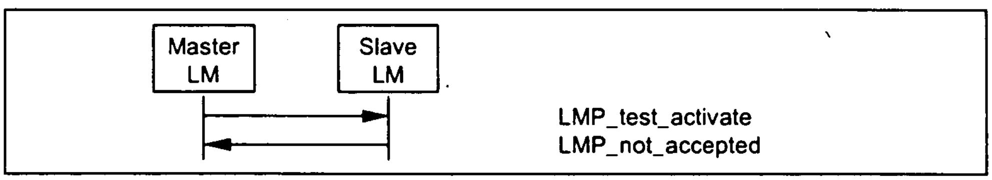
LMP has PDUs to support different Bluetooth test modes, which are used for certification and compliance testing of the Bluetooth radio and baseband. See "Bluetooth Test Mode" on page 803 for a detailed description of these test modes.

6.1 ACTIVATION AND DEACTIVATION OF TEST MODE

The test mode is activated by sending LMP_test_activate to the device under test (DUT). The DUT is always the slave. The link manager must be able to receive this message anytime. If entering test mode is locally enabled in the DUT it responds with LMP_accepted and test mode is entered. Otherwise the DUT responds with LMP_not_accepted and the DUT remains in normal operation. The reason code in LMP_not_accepted shall be *PDU not allowed.*



Sequence 60: Activation of test mode successful.

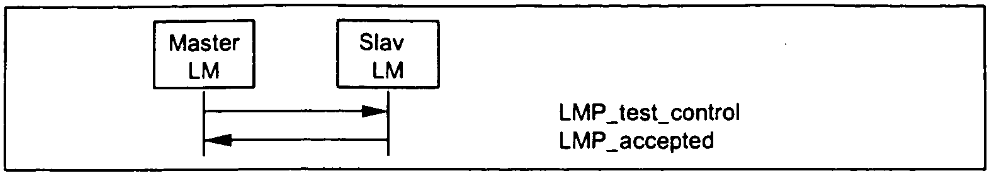


Sequence 61: Activation of test mode fails. Slave is not allowed to enter test mode.

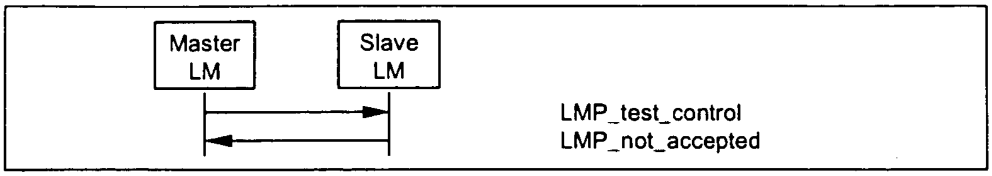
The test mode can be deactivated in two ways. Sending LMP_test_control with the test scenario set to "exit test mode" exits the test mode and the slave returns to normal operation still connected to the master. Sending LMP_detach to the DUT ends the test mode and the connection.

6.2 CONTROL OF TEST MODE

When the DUT has entered test mode, the PDU LMP_test_control can be sent to the DUT to start a specific test. This PDU is acknowledged with LMP_accepted. If a device that is not in test mode receives LMP_test_control it responds with LMP_not_accepted, where the reason code shall be *PDU not allowed.*



Sequence 62: Control of test mode successful.



Sequence 63: Control of test mode rejected since slave is not in test mode.

6.3 SUMMARY OF TEST MODE PDUs

The PDUs used for test purposes are summarized in the following table. For a detailed description of the parameters, see Bluetooth Test Mode Table 3.2 on page 817.

M/O	LMP PDU	Length	op code	Packet type	Possible direction	Contents	Position in payload
M	LMP_test_activate	1	56	DM1/DV	m → s	-	
M	LMP_test_control	10	57	DM1	m → s	test scenario	2
						hopping mode	3
						TX frequency	4
						RX frequency	5
						power control mode	6
						poll period	7
						packet type	8
						length of test data	9-10

Table 6.1: Test mode PDUs.

7 ERROR HANDLING

If the Link Manager receives a PDU with unrecognized opcode, it responds with LMP_not_accepted with the reason code *unknown LMP PDU.* The opcode parameter that is echoed back is the unrecognized opcode.

If the Link Manager receives a PDU with invalid parameters, it responds with LMP_not_accepted with the reason code *invalid LMP parameters.*

If the maximum response time, see Section 1 on page 191, is exceeded or if a link loss is detected (see Baseband Specification Section 10.11, on page 126), the party that waits for the response shall conclude that the procedure has terminated unsuccessfully.

Erroneous LMP messages can be caused by errors on the channel or systematic errors at the transmit side. To detect the latter case, the LM should monitor the number of erroneous messages and disconnect if it exceeds a threshold, which is implementation-dependent.

Since LMP PDUs are not interpreted in real time, collision situations can occur where both LMs initiate the same procedure and both cannot be completed. In this situation, the master shall reject the slave-initiated procedure by sending LMP_not_accepted with the reason code 'LMP Error Transaction Collision'. The master-initiated procedure shall then be completed.

8 LIST OF FIGURES

9 LIST OF TABLES

Part D

LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION

This document describes the Bluetooth logical link control and adaptation protocol (L2CAP). This protocol supports higher level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information. This document is part of the Bluetooth Specification. This document describes the protocol state machine, packet format and composition, and a test interface required for the Bluetooth test and certification program.

Bluetooth.

CONTENTS

1 INTRODUCTION

This section of the Bluetooth Specification defines the Logical Link Control and Adaptation Layer Protocol, referred to as L2CAP. L2CAP is layered over the Baseband Protocol and resides in the data link layer as shown in Figure 1.1. L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation, and group abstractions. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kilobytes in length.

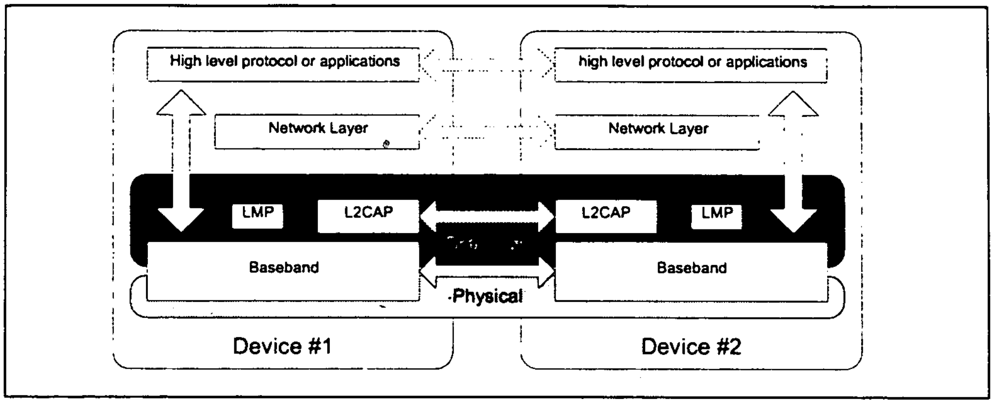


Figure 1.1: L2CAP within protocol layers

The “Baseband Specification” on page 33 defines two link types: Synchronous Connection-Oriented (SCO) links and Asynchronous Connection-Less (ACL) links..SCO links support real-time voice traffic using reserved bandwidth. ACL links support best effort traffic. The L2CAP Specification is defined for only ACL links and no support for SCO links is planned.

For ACL links, use of the AUX1 packet on the ACL link is prohibited. This packet type supports no data integrity checks (no CRC). Because L2CAP depends on integrity checks in the Baseband to protect the transmitted information, AUX1 packets must never be used to transport L2CAP packets.

The format of the ACL payload header is shown below. Figure 1.2 on page 250 displays the payload header used for single-slot packets and Figure 1.3 displays the header used in multi-slot packets. The only difference is the size of the length field. The packet type (a field in the Baseband header) distinguishes single-slot packets from multi-slot packets.

LSB			MSB
2	1	5	
L_CH	FLOW	LENGTH	

Figure 1.2: ACL Payload Header for single-slot packets

LSB			MSB
2	1	9	4
L_CH	FLOW	LENGTH	undefined

Figure 1.3: ACL Payload Header for multi-slot packets

The 2-bit logical channel (L_CH) field, defined in Table 1.1, distinguishes L2CAP packets from Link Manager Protocol (page 185) packets. The remaining code is reserved for future use.

L_CH code	Logical Channel	Information
00	RESERVED	Reserved for future use
01	L2CAP	Continuation of L2CAP packet
10	L2CAP	Start of L2CAP packet
11	LMP	Link Manager Protocol

Table 1.1: Logical channel L_CH field contents

The FLOW bit in the ACL header is managed by the Link Controller (LC), a Baseband implementation entity, and is normally set to1 ('flow on'). It is set to 0 ('flow off') when no further L2CAP traffic shall be sent over the ACL link. Sending an L2CAP packet with the FLOW bit set to 1 resumes the flow of incoming L2CAP packets. This is described in more detail in "Baseband Specification" on page 33.

1.1 L2CAP FUNCTIONAL REQUIREMENTS

The functional requirements for L2CAP include protocol multiplexing, segmentation and reassembly (SAR), and group management. Figure 1.4 illustrates how L2CAP fits into the Bluetooth Protocol Stack. L2CAP lies above the Baseband Protocol (page 33) and interfaces with other communication protocols such as the Bluetooth Service Discovery Protocol (SDP, page 323), RFCOMM (page 385), and Telephony Control (TCS, page 429). Voice-quality channels for audio and telephony applications are usually run over Baseband SCO links. Packetized audio data, such as IP Telephony, may be sent using communication protocols running over L2CAP.

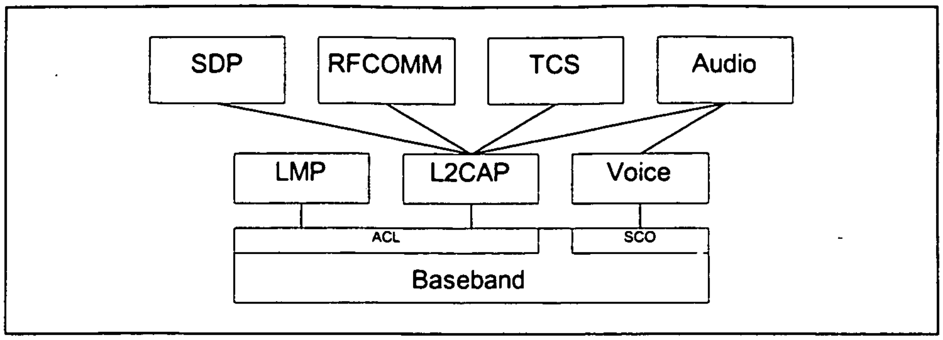


Figure 1.4: L2CAP in Bluetooth Protocol Architecture

Essential protocol requirements for L2CAP include simplicity and low overhead. Implementations of L2CAP must be applicable for devices with limited computational resources. L2CAP should not consume excessive power since that significantly sacrifices power efficiency achieved by the Bluetooth Radio. Memory requirements for protocol implementation should also be kept to a minimum.

The protocol complexity should be acceptable to personal computers, PDAs, digital cellular phones, wireless headsets, joysticks and other wireless devices supported by Bluetooth. Furthermore, the protocol should be designed to achieve reasonably high bandwidth efficiency.

- *Protocol Multiplexing*

  L2CAP must support protocol multiplexing because the Baseband Protocol does not support any 'type' field identifying the higher layer protocol being multiplexed above it. L2CAP must be able to distinguish between upper layer protocols such as the Service Discovery Protocol (page 323), RFCOMM (page 385), and Telephony Control (page 429).

- *Segmentation and Reassembly*

  Compared to other wired physical media, the data packets defined by the Baseband Protocol (page 33) are limited in size. Exporting a maximum transmission unit (MTU) associated with the largest Baseband payload (341 bytes for DH5 packets) limits the efficient use of bandwidth for higher layer protocols that are designed to use larger packets. Large L2CAP packets must be segmented into multiple smaller Baseband packets prior to their transmission over the air. Similarly, multiple received Baseband packets may be reassembled into a single larger L2CAP packet following a simple integrity check (described in Section 2.4.2 on page 256). The Segmentation and Reassembly (SAR) functionality is absolutely necessary to support protocols using packets larger than those supported by the Baseband.

- *Quality of Service*

  The L2CAP connection establishment process allows the exchange of information regarding the quality of service (QoS) expected between two Blue-

tooth units. Each L2CAP implementation must monitor the resources used by the protocol and ensure that QoS contracts are honoured.

- *Groups*

  Many protocols include the concept of a group of addresses. The Baseband Protocol supports the concept of a piconet, a group of devices synchronously hopping together using the same clock. The L2CAP group abstraction permits implementations to efficiently map protocol groups on to piconets. Without a group abstraction, higher level protocols would need to be exposed to the Baseband Protocol and Link Manager functionality in order to manage groups efficiently.

1.2 ASSUMPTIONS

The protocol is designed based on the following assumptions:

1. The ACL link between two units is set up using the Link Manager Protocol (page 185). The Baseband provides orderly delivery of data packets, although there might be individual packet corruption and duplicates. No more than 1 ACL link exists between any two devices.
2. The Baseband always provides the impression of full-duplex communication channels. This does not imply that all L2CAP communications are bi-directional. Multicasts and unidirectional traffic (e.g., video) do not require duplex channels.
3. L2CAP provides a reliable channel using the mechanisms available at the Baseband layer. The Baseband always performs data integrity checks when requested and resends data until it has been successfully acknowledged or a timeout occurs. Because acknowledgements may be lost, timeouts may occur even after the data has been successfully sent. The Baseband protocol uses a 1-bit sequence number that removes duplicates. Note that the use of Baseband broadcast packets is prohibited if reliability is required since all broadcasts start the first segment of an L2CAP packet with the same sequence bit.

1.3 SCOPE

The following features are outside the scope of L2CAP's responsibilities:

- L2CAP does not transport audio designated for SCO links.
- L2CAP does not enforce a reliable channel or ensure data integrity, that is, L2CAP performs no retransmissions or checksum calculations.
- L2CAP does not support a reliable multicast channel. See Section 4.2.
- L2CAP does not support the concept of a global group name.

2 GENERAL OPERATION

The Logical Link Control and Adaptation Protocol (L2CAP) is based around the concept of *'channels'*. Each one of the end-points of an L2CAP channel is referred to by a *channel identifier.*

2.1 CHANNEL IDENTIFIERS

Channel identifiers (CIDs) are local names representing a logical channel end-point on the device. Identifiers from 0x0001 to 0x003F are reserved for specific L2CAP functions. The null identifier (0x0000) is defined as an illegal identifier and must never be used as a destination end-point. Implementations are free to manage the remaining CIDs in a manner best suited for that particular implementation, with the provision that the same CID is not reused as a local L2CAP channel endpoint for multiple simultaneous L2CAP channels between a local device and some remote device. Table 2.1 summarizes the definition and partitioning of the CID name space.

CID assignment is relative to a particular device and a device can assign CIDs independently from other devices (unless it needs to use any of the reserved CIDs shown in the table below). Thus, even if the same CID value has been assigned to (remote) channel endpoints by several remote devices connected to a single local device, the local device can still uniquely associate each remote CID with a different device.

CID	Description
0x0000	Null identifier
0x0001	Signalling channel
0x0002	Connectionless reception channel
0x0003-0x003F	Reserved
0x0040-0xFFFF	Dynamically allocated

Table 2.1: CID Definitions

2.2 OPERATION BETWEEN DEVICES

Figure 2.1 on page 254 illustrates the use of CIDs in a communication between corresponding peer L2CAP entities in separate devices. The connection-oriented data channels represent a connection between two devices, where a CID identifies each endpoint of the channel. The connectionless channels restrict data flow to a single direction. These channels are used to support a channel 'group' where the CID on the source represents one or more remote devices. There are also a number of CIDs reserved for special purposes. The signalling channel is one example of a reserved channel. This channel is used to create and establish connection-oriented data channels and to negotiate changes in the characteristics of these channels. Support for a signalling chan-

nel within an L2CAP entity is mandatory. Another CID is reserved for all incoming connectionless data traffic. In the example below, a CID is used to represent a group consisting of device #3 and #4. Traffic sent from this channel ID is directed to the remote channel reserved for connectionless data traffic.

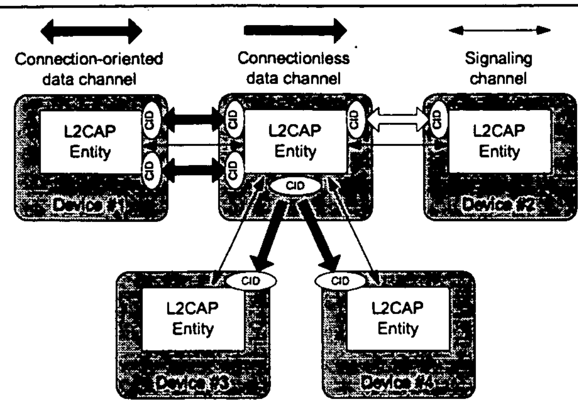


Figure 2.1: Channels between devices

Table 2.2 describes the various channels and their source and destination identifiers. An 'allocated' channel is created to represent the local endpoint and should be in the range 0x0040 to 0xFFFF. Section 3 on page 258 describes the state machine associated with each connectionless channel. Section 4.1 on page 272 describes the packet format associated with bi-directional channels and Section 4.2 on page 273 describes the packet format associated with uni-directional channels.

Channel Type	Local CID	Remote CID
Connection-oriented	Dynamically allocated	Dynamically allocated
Connectionless data	Dynamically allocated	0x0002 (fixed)
Signalling	0x0001 (fixed)	0x0001 (fixed)

Table 2.2: Types of Channel Identifiers

2.3 OPERATION BETWEEN LAYERS

L2CAP implementations should follow the general architecture described below. L2CAP implementations must transfer data between higher layer protocols and the lower layer protocol. This document lists a number of services that should be exported by any L2CAP implementation. Each implementation must also support a set of signalling commands for use between L2CAP implementations. L2CAP implementations should also be prepared to accept certain types of events from lower layers and generate events to upper layers. How these events are passed between layers is an implementation-dependent process.

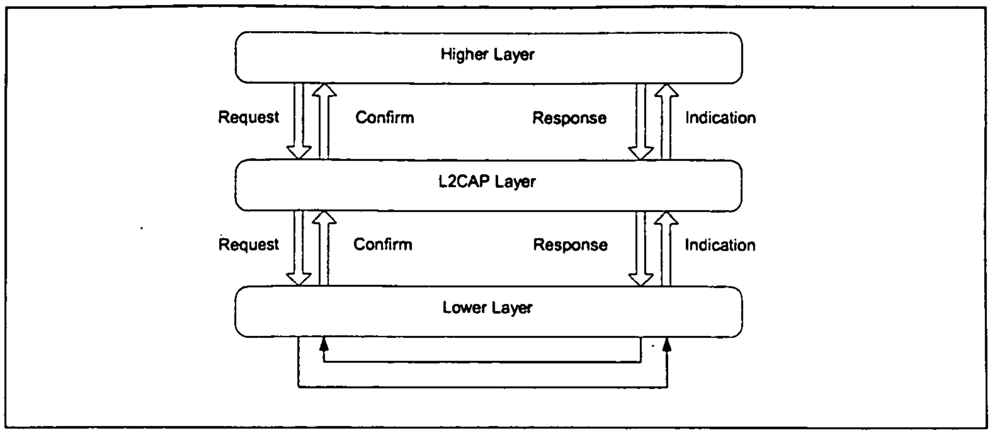


Figure 2.2: L2CAP Architecture

2.4 SEGMENTATION AND REASSEMBLY

Segmentation and reassembly (SAR) operations are used to improve efficiency by supporting a maximum transmission unit (MTU) size larger than the largest Baseband packet. This reduces overhead by spreading the network and transport packets used by higher layer protocols over several Baseband packets. All L2CAP packets may be segmented for transfer over Baseband packets. The protocol does not perform any segmentation and reassembly operations but the packet format supports adaptation to smaller physical frame sizes. An L2CAP implementation exposes the outgoing (i.e., the remote host's receiving) MTU and segments higher layer packets into 'chunks' that can be passed to the Link Manager via the Host Controller Interface (HCI), whenever one exists. On the receiving side, an L2CAP implementation receives 'chunks' from the HCI and reassembles those chunks into L2CAP packets using information provided through the HCI and from the packet header.

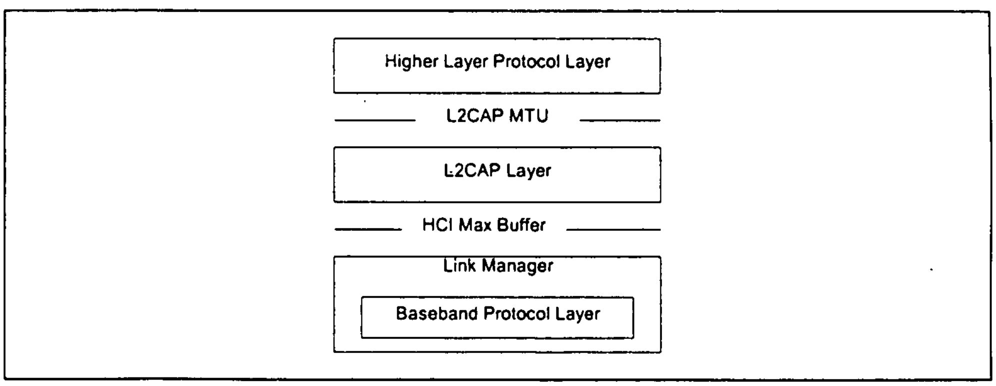


Figure 2.3: L2CAP SAR Variables

Segmentation and Reassembly is implemented using very little overhead in Baseband packets. The two L_CH bits defined in the first byte of Baseband

payload (also called the frame header) are used to signal the start and continuation of L2CAP packets. L_CH shall be '10' for the first segment in an L2CAP packet and '01' for a continuation segment. An example use of SAR is shown in Figure 2.4.

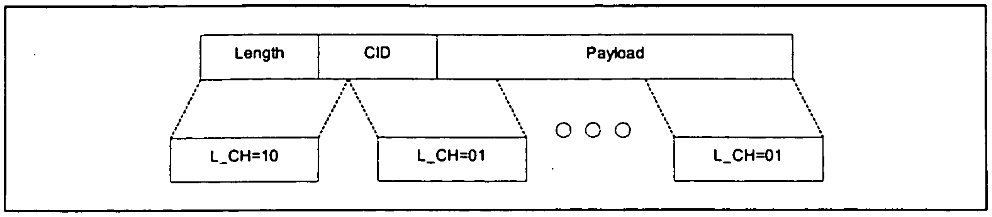


Figure 2.4: L2CAP segmentation

2.4.1 Segmentation Procedures

The L2CAP maximum transmission unit (MTU) will be exported using an implementation specific service interface. It is the responsibility of the higher layer protocol to limit the size of packets sent to the L2CAP layer below the MTU limit. An L2CAP implementation will segment the packet into protocol data units (PDUs) to send to the lower layer. If L2CAP runs directly over the Baseband Protocol, an implementation may segment the packet into Baseband packets for transmission over the air. If L2CAP runs above the host controller interface (typical scenario), an implementation may send block-sized chunks to the host controller where they will be converted into Baseband packets. All L2CAP segments associated with an L2CAP packet must be passed through to the Baseband before any other L2CAP packet destined to the same unit may be sent.

2.4.2 Reassembly Procedures

The Baseband Protocol delivers ACL packets in sequence and protects the integrity of the data using a 16-bit CRC. The Baseband also supports reliable connections using an automatic repeat request (ARQ) mechanism. As the Baseband controller receives ACL packets, it either signals the L2CAP layer on the arrival of each Baseband packets, or accumulates a number of packets before the receive buffer fills up or a timer expires before signalling the L2CAP layer.

L2CAP implementations must use the length field in the header of L2CAP packets, see Section 4 on page 272, as a consistency check and discard any L2CAP packets that fail to match the length field. If channel reliability is not needed, packets with improper lengths may be silently discarded. For reliable channels, L2CAP implementations must indicate to the upper layer that the channel has become unreliable. Reliable channels are defined by having an infinite flush timeout value as specified in Section 6.2 on page 290.

Figure 2.5 on page 257 illustrates the use of segmentation and reassembly operations to transmit a single higher layer PDU. Note that while there is a one-to-one mapping between a high layer PDU and an L2CAP packet, the segment

size used by the segmentation and reassembly routines is left to the implementation and may differ from the sender to the receiver.

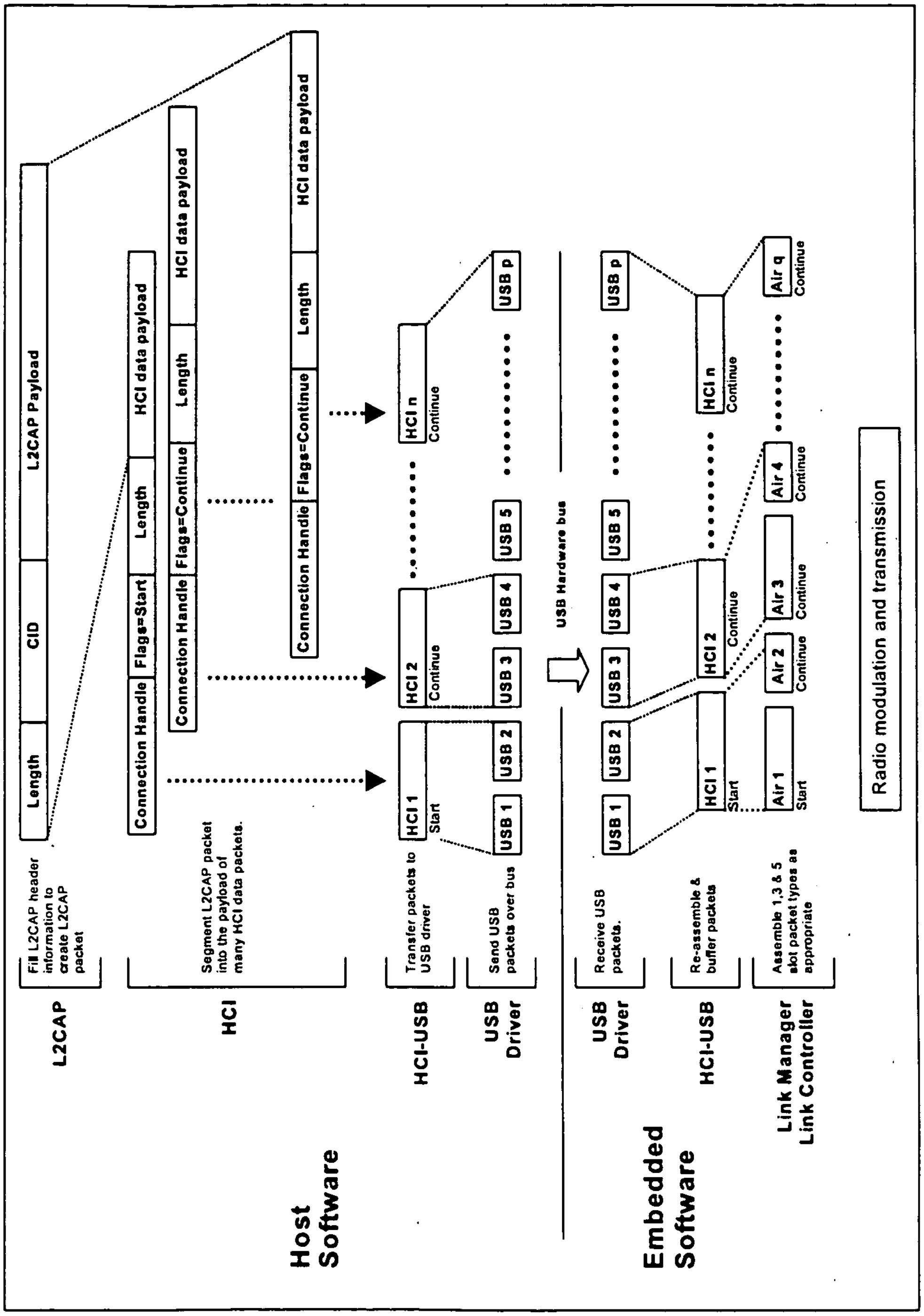


Figure 2.5: Segmentation and Reassembly Services in a unit with an HCI[1]

3 STATE MACHINE

This section describes the L2CAP connection-oriented channel state machine. The section defines the states, the events causing state transitions, and the actions to be performed in response to events. This state machine is only pertinent to bi-directional CIDs and is not representative of the signalling channel or the uni-directional channel.

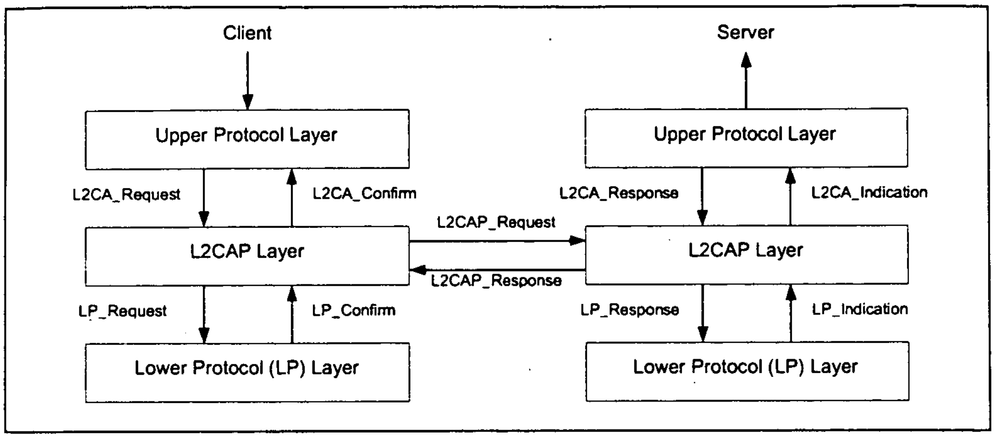


Figure 3.1: L2CAP Layer Interactions

Figure 3.1 illustrates the events and actions performed by an implementation of the L2CAP layer. Client and Server simply represent the initiator of the request and the acceptor of the request respectively. An application-level Client would both initiate and accept requests. The naming convention is as follows. The interface between two layers (vertical interface) uses the prefix of the lower layer offering the service to the higher layer, e.g., L2CA. The interface between two entities of the same layer (horizontal interface) uses the prefix of the protocol (adding a P to the layer identification), e.g., L2CAP. Events coming from above are called Requests (Req) and the corresponding replies are called Confirms (Cfm). Events coming from below are called Indications (Ind) and the corresponding replies are called Responses (Rsp). Responses requiring further processing are called Pending (Pnd). The notation for Confirms and Responses assumes positive replies. Negative replies are denoted by a 'Neg' suffix such as L2CAP_ConnectCfmNeg.

While Requests for an action always result in a corresponding Confirmation (for the successful or unsuccessful satisfaction of the action), Indications do not always result into corresponding Responses. The latter is especially true, if the Indications are informative about locally triggered events, e.g., seeing the

1. For simplicity, the stripping of any additional HCI and USB specific information fields prior to the creation of the baseband packets (Air_1, Air_2, etc.) is not shown in the figure.

LP_QoSViolationInd in Section 3.1.1 on page 259, or *L2CA_TimeOutInd* in Section 3.2.4 on page 264.

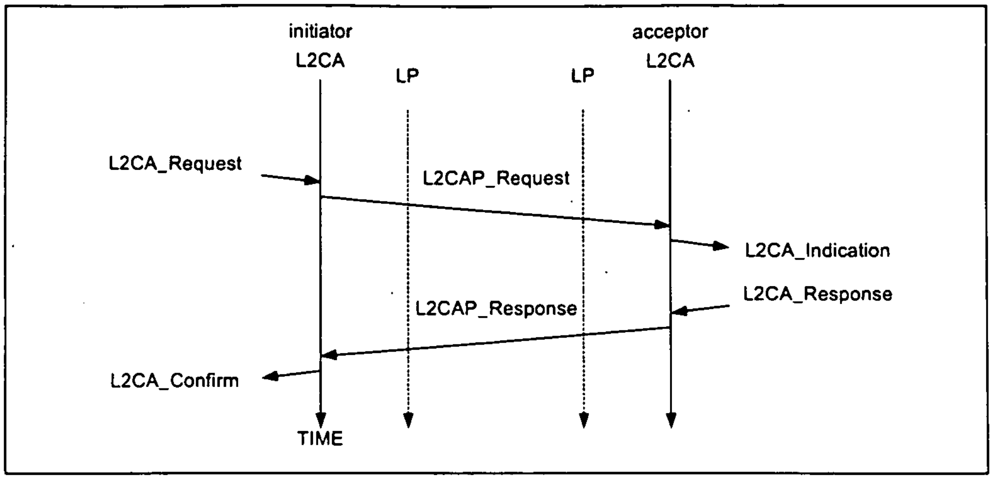


Figure 3.2: MSC of Layer Interactions

Figure 3.2 uses a message sequence chart (MSC) to illustrate the normal sequence of events. The two outer vertical lines represent the L2CA interface on the initiator (the device issuing a request) and the acceptor (the device responding to the initiator's request). Request commands at the L2CA interface result in Requests defined by the protocol. When the protocol communicates the request to the acceptor, the remote L2CA entity presents the upper protocol with an Indication. When the acceptor's upper protocol responds, the response is packaged by the protocol and communicated back the to initiator. The result is passed back to the initiator's upper protocol using a Confirm message.

3.1 EVENTS

Events are all incoming messages to the L2CA layer along with timeouts. Events are partitioned into five categories: Indications and Confirms from lower layers, Requests and Responses from higher layers, data from peers, signal Requests and Responses from peers, and events caused by timer expirations.

3.1.1 Lower-Layer Protocol (LP) to L2CAP events

- *LP_ConnectCfm*

  Confirms the request (see *LP_ConnectReq* in Section 3.2.1) to establish a lower layer (Baseband) connection. This includes passing the authentication challenge if authentication is required to establish the physical link.

- *LP_ConnectCfmNeg*

  Confirms the failure of the request (see *LP_ConnectReq* in Section 3.2.1) to establish a lower layer (Baseband) connection failed. This could be because

the device could not be contacted, refused the request, or the LMP authentication challenge failed.

- *LP_ConnectInd*

  Indicates the lower protocol has successfully established connection. In the case of the Baseband, this will be an ACL link. An L2CAP entity may use to information to keep track of what physical links exist.

- *LP_DisconnectInd*

  Indicates the lower protocol (Baseband) has been shut down by LMP commands or a timeout event.

- *LP_QoSCfm*

  Confirms the request (see *LP_QoSReq* in Section 3.2.1) for a given quality of service.

- *LP_QoSCfmNeg*

  Confirms the failure of the request (see *LP_QoSReq* in Section 3.2.1) for a given quality of service.

- *LP_QoSViolationInd*

  Indicates the lower protocol has detected a violation of the QoS agreement specified in the previous *LP_QoSReq* (see Section 3.2.1).

3.1.2 L2CAP to L2CAP Signalling events

L2CAP to L2CAP signalling events are generated by each L2CAP entity following the exchange of the corresponding L2CAP signalling PDUs, see Section 5. L2CAP signalling PDUs, like any other L2CAP PDUs, are received from a lower layer via a lower protocol indication event. For simplicity of the presentation, we avoid a detailed description of this process, and we assume that signalling events are exchanged directly between the L2CAP peer entities as shown in Figure 3.1 on page 258.

- *L2CAP_ConnectReq*

  A Connection Request packet has been received.

- *L2CAP_ConnectRsp*

  A Connection Response packet has been received with a positive result indicating that the connection has been established.

- *L2CAP_ConnectRspPnd*

  A Connection Response packet has been received indicating the remote endpoint has received the request and is processing it.

- *L2CAP_ConnectRspNeg*

  A Connection Response packet has been received, indicating that the connection could not be established.

- *L2CAP_ConfigReq*

A Configuration Request packet has been received indicating the remote endpoint wishes to engage in negotiations concerning channel parameters.

- *L2CAP_ConfigRsp*

  A Configuration Response packet has been received indicating the remote endpoint agrees with all the parameters being negotiated.
- *L2CAP_ConfigRspNeg*

  A Configuration Response packet has been received indicating the remote endpoint does not agree to the parameters received in the response packet.
- *L2CAP_DisconnectReq*

  A Disconnection Request packet has been received and the channel must initiate the disconnection process. Following the completion of an L2CAP channel disconnection process, an L2CAP entity should return the corresponding local CID to the pool of 'unassigned' CIDs.
- *L2CAP_DisconnectRsp*

  A Disconnection Response packet has been received. Following the receipt of this signal, the receiving L2CAP entity may return the corresponding local CID to the pool of unassigned CIDs. There is no corresponding negative response because the Disconnect Request must succeed.

3.1.3 L2CAP to L2CAP Data events

- *L2CAP_Data*

  A Data packet has been received.

3.1.4 Upper-Layer to L2CAP events

- *L2CA_ConnectReq*

  Request from upper layer for the creation of a channel to a remote device.
- *L2CA_ConnectRsp*

  Response from upper layer to the indication of a connection request from a remote device (see L2CA_ConnectInd in Section 3.2.4).
- *L2CA_ConnectRspNeg*

  Negative response (rejection) from upper layer to the indication of a connection request from a remote device (see *L2CA_ConnectInd* in Section 3.2.4).
- *L2CA_ConfigReq*

  Request from upper layer to (re)configure the channel.
- *L2CA_ConfigRsp*

  Response from upper layer to the indication of a (re) configuration request (see *L2CA_ConfigInd* in Section 3.2.4).
- *L2CA_ConfigRspNeg*

A negative response from upper layer to the indication of a (re) configuration request (see *L2CA_ConfigInd* in Section 3.2.4).

- *L2CA_DisconnectReq*

  Request from upper layer for the immediate disconnection of a channel.

- *L2CA_DisconnectRsp*

  Response from upper layer to the indication of a disconnection request (see *L2CA_DisconnectInd* in Section 3.2.4). There is no corresponding negative response, the disconnect indication must always be accepted.

- *L2CA_DataRead*

  Request from upper layer for the transfer of received data from L2CAP entity to upper layer.

- *L2CA_DataWrite*

  Request from upper layer for the transfer of data from the upper layer to L2CAP entity for transmission over an open channel.

3.1.5 Timer events

- *RTX*

  The Response Timeout eXpired (RTX) timer is used to terminate the channel when the remote endpoint is unresponsive to signalling requests. This timer is started when a signalling request (see Section 5 on page 275) is sent to the remote device. This timer is disabled when the response is received. If the initial timer expires, a duplicate Request message may be sent or the channel identified in the request may be disconnected. If a duplicate Request message is sent, the RTX timeout value must be reset to a new value at least double the previous value.

  Implementations have the responsibility to decide on the maximum number of Request retransmissions performed at the L2CAP level before disconnecting the channel. The decision should be based on the flush timeout of the signalling link. The longer the flush timeout, the more retransmissions may be performed at the physical layer and the reliability of the channel improves, requiring fewer retransmissions at the L2CAP level. For example, if the flush timeout is infinite, no retransmissions should be performed at the L2CAP level.

  The value of this timer is implementation-dependent but the minimum initial value is 1 second and the maximum initial value is 60 seconds. One RTX timer MUST exist for each outstanding signalling request, including each Echo Request. The timer disappears on the final expiration, when the response is received, or the physical link is lost. The maximum elapsed time between the initial start of this timer and the initiation of channel disconnection (if no response is received) is 60 seconds.

- *ERTX*

  The Extended Response Timeout eXpired (ERTX) timer is used in place of the RTX timer when it is suspected the remote endpoint is performing addi-

tional processing of a request signal. This timer is started when the remote endpoint responds that a request is pending, e.g., when an *L2CAP_ConnectRspPnd* event is received. This timer is disabled when the formal response is received or the physical link is lost. If the initial timer expires, a duplicate Request may be sent or the channel may be disconnected. If a duplicate Request is sent, the particular ERTX timer disappears, replaced by a new RTX timer and the whole timing procedure restarts as described previously for the RTX timer.

The value of this timer is implementation-dependent but the minimum initial value is 60 seconds and the maximum initial value is 300 seconds. Similar to RTX, there MUST be at least one ERTX timer for each outstanding request that received a Pending response. There should be at most one (RTX or ERTX) associated with each outstanding request. The maximum elapsed time between the initial start of this timer and the initiation of channel disconnection (if no response is received) is 300 seconds.

3.2 ACTIONS

Actions are partitioned into five categories: Confirms and Indications to higher layers, Request and Responses to lower layers, Requests and Responses to peers, data transmission to peers, and setting timers.

3.2.1 L2CAP to Lower Layer actions

- *LP_ConnectReq*

  L2CAP requests the lower protocol to create a connection. If a physical link to the remote device does not exist, this message must be sent to the lower protocol to establish the physical connection. Since no more than a single ACL link between two devices is assumed, see Section 1.2 on page 252, additional L2CAP channels between these two devices must share the same baseband ACL link.
  Following the processing of the request, the lower layer returns with an *LP_ConnectCfm* or an *LP_ConnectCfmNeg* to indicate whether the request has been satisfied or not, respectively.

- *LP_QoSReq*

  L2CAP requests the lower protocol to accommodate a particular QoS parameter set. Following the processing of the request, the lower layer returns with an *LP_QoSCfm* or an *LP_QoSCfmNeg* to indicate whether the request has been satisfied or not, respectively

- *LP_ConnectRsp*

  A positive response accepting the previous connection indication request (see *LP_ConnectInd* in Section 3.1.1).

- *LP_ConnectRspNeg*

  A negative response denying the previous connection indication request (see *LP_ConnectInd* in Section 3.1.1).

3.2.2 L2CAP to L2CAP Signalling actions

This section contains the same names identified in Section 3.1.2 except the actions refer to the transmission, rather than reception, of these messages.

3.2.3 L2CAP to L2CAP Data actions

This section is the counterpart of Section 3.1.3. Data transmission is the action performed here.

3.2.4 L2CAP to Upper Layer actions

- *L2CA_ConnectInd*

  Indicates a Connection Request has been received from a remote device (see *L2CA_ConnectReq* in Section 3.1.4).

- *L2CA_ConnectCfm*

  Confirms that a Connection Request has been accepted (see (*L2CAP_ConnectReq* in Section 3.1.4) following the receipt of a Connection message from the remote device.

- *L2CA_ConnectCfmNeg*

  Negative confirmation (failure) of a Connection Request (see *L2CA_ConnectReq* in Section 3.1.4). An RTX timer expiration (see Section 3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Connect Request can substitute for a negative Connect Response and result in this action.

- *L2CA_ConnectPnd*

  Confirms that a Connection Response (pending) has been received from the remote device.

- *L2CA_ConfigInd*

  Indicates a Configuration Request has been received from a remote device.

- *L2CA_ConfigCfm*

  Confirms that a Configuration Request has been accepted (see *L2CA_ConfigReq* in Section 3.1.4) following the receipt of a Configuration Response from the remote device.

- *L2CA_ConfigCfmNeg*

  Negative confirmation (failure) of a Configuration Request (see *L2CA_ConfigReq* in Section 3.1.4). An RTX timer expiration (see Section 3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Connect Request can substitute for a negative Connect Response and result in this action.

- *L2CA_DisconnectInd*

  Indicates a Disconnection Request has been received from a remote device or the remote device has been disconnected because it has failed to respond to a signalling request. See Section 3.1.5

- *L2CA_DisconnectCfm*

  Confirms that a Disconnect Request has been processed by the remote device (see *L2CA_DisconnectReq* in Section 3.1.4) following the receipt of a Disconnection Response from the remote device. An RTX timer expiration (see Section 3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Disconnect Request can substitute for a Disconnect Response and result in this action. Upon receiving this event the upper layer knows the L2CAP channel has been terminated. There is no corresponding negative confirm.

- *L2CA_TimeOutInd*

  Indicates that a RTX or ERTX timer has expired. This indication will occur an implementation-dependant number of times before the L2CAP implementation will give up and send a L2CA_DisconnectInd.

- *L2CA_QoSViolationInd*

  Indicates that the quality of service agreement has been violated.

3.3 CHANNEL OPERATIONAL STATES

- *CLOSED*

  In this state, there is no channel associated with this CID. This is the only state when a link level connection (Baseband) may not exist. Link disconnection forces all other states into the CLOSED state.

- *W4_L2CAP_CONNECT_RSP*

  In this state, the CID represents a local end-point and an L2CAP_ConnectReq message has been sent referencing this endpoint and it is now waiting for the corresponding L2CAP_ConnectRsp message.

- *W4_L2CA_CONNECT_RSP*

  In this state, the remote end-point exists and an *L2CAP_ConnectReq* has been received by the local L2CAP entity. An L2CA_ConnectInd has been sent to the upper layer and the part of the local L2CAP entity processing the received *L2CAP_ConnectReq* waits for the corresponding response. The response may require a security check to be performed.

- *CONFIG*

  In this state, the connection has been established but both sides are still negotiating the channel parameters. The Configuration state may also be entered when the channel parameters are being renegotiated. Prior to entering the CONFIG state, all outgoing data traffic should be suspended since the traffic parameters of the data traffic are to be renegotiated. Incoming data traffic must be accepted until the remote channel endpoint has entered the CONFIG state.

In the CONFIG state, both sides must issue L2CAP_ConfigReq messages – if only defaults are being used, a null message should be sent, see Section 5.4 on page 280. If a large amount of parameters need to be negotiated, multiple messages may be sent to avoid any MTU limitations and negotiate incrementally – see Section 6 on page 289 for more details.

Moving from the CONFIG state to the OPEN state requires both sides to be ready. An L2CAP entity is ready when it has received a positive response to its final request and it has positively responded to the final request from the remote device.

- *OPEN*

  In this state, the connection has been established and configured, and data flow may proceed.

- *W4_L2CAP_DISCONNECT_RSP*

  In this state, the connection is shutting down and an L2CAP_DisconnectReq message has been sent. This state is now waiting for the corresponding response.

- *W4_L2CA_DISCONNECT_RSP*

  In this state, the connection on the remote endpoint is shutting down and an L2CAP_DisconnectReq message has been received. An L2CA_DisconnectInd has been sent to the upper layer to notify the owner of the CID that the remote endpoint is being closed. This state is now waiting for the corresponding response from the upper layer before responding to the remote endpoint.

3.4 MAPPING EVENTS TO ACTIONS

Table 3.1 defines the actions taken in response to events that occur in a particular state. Events that are not listed in the table, nor have actions marked N/C (for no change), are assumed to be errors and silently discarded.

Data input and output events are only defined for the Open and Configuration states. Data may not be received during the initial Configuration state, but may be received when the Configuration state is re-entered due to a reconfiguration process. Data received during any other state should be silently discarded.

Event	Current State	Action	New State
LP_ConnectCfm	CLOSED	Flag physical link as up and initiate the L2CAP connection.	CLOSED
LP_ConnectCfmNeg	CLOSED	Flag physical link as down and fail any outstanding service connection requests by sending an L2CA_ConnectCfmNeg message to the upper layer.	CLOSED
LP_ConnectInd	CLOSED	Flag link as up.	CLOSED
LP_DisconnectInd	CLOSED	Flag link as down.	CLOSED
LP_DisconnectInd	Any except CLOSED	Send upper layer L2CA_DisconnectInd message.	CLOSED
LP_QoSViolationInd	Any but OPEN	Discard	N/C
LP_QoSViolationInd	OPEN	Send upper layer L2CA_QoSViolationInd message. If service level is guaranteed, terminate the channel.	OPEN or W4_L2CA_DISCONNECT_RSP
L2CAP_ConnectReq	CLOSED. (CID dynamically allocated from free pool.)	Send upper layer L2CA_ConnectInd. Optionally: Send peer L2CAP_ConnectRspPnd	W4_L2CA_CONNECT_RSP
L2CAP_ConnectRsp	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectCfm message. Disable RTX timer.	CONFIG
L2CAP_ConnectRspPnd	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectPnd message. Disable RTX timer and start ERTX timer.	N/C
L2CAP_ConnectRspNeg	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectCfmNeg message. Return CID to free pool. Disable RTX/ERTX timers.	CLOSED
L2CAP_ConfigReq	CLOSED	Send peer L2CAP_ConfigRspNeg message.	N/C
L2CAP_ConfigReq	CONFIG	Send upper layer L2CA_ConfigInd message.	N/C

Table 3.1: L2CAP Channel State Machine

Event	Current State	Action	New State
L2CAP_ConfigReq	OPEN	Suspend data transmission at a convenient point. Send upper layer L2CA_ConfigInd message.	CONFIG
L2CAP_ConfigRsp	CONFIG	Send upper layer L2CA_ConfigCfm message. Disable RTX timer. If an L2CAP_ConfigReq message has been received and positively responded to, then enter OPEN state, otherwise remain in CONFIG state.	N/C or OPEN
L2CAP_ConfigRsp Neg	CONFIG	Send upper layer L2CA_ConfigCfmNeg message. Disable RTX timer.	N/C
L2CAP_Disconnect Req	CLOSED	Send peer L2CAP_DisconnectRsp message.	N/C
L2CAP_Disconnect Req	Any except CLOSED	Send upper layer L2CA_DisconnectInd message.	W4_L2CA_DISCONNECT_RSP
L2CAP_Disconnect Rsp	W4_L2CAP_DISCONNECT_RSP	Send upper layer L2CA_DisconnectCfm message. Disable RTX timer.	CLOSED
L2CAP_Data	OPEN or CONFIG	If complete L2CAP packet received, send upper layer L2CA_Read confirm.	N/C
L2CA_ConnectReq	CLOSED (CID dynamically allocated from free pool)	Send peer LP2CAP_ConnectReq message. Start RTX timer.	W4_L2CAP_CONNECT_RSP
L2CA_ConnectRsp	W4_L2CA_CONNECT_RSP	Send peer L2CAP_ConnectRsp message.	CONFIG
L2CA_ConnectRsp Neg	W4_L2CA_CONNECT_RSP	Send peer L2CAP_ConnectRspNeg message. Return CID to free pool.	CLOSED
L2CA_ConfigReq	CLOSED	Send upper layer L2CA_ConfigCfmNeg message.	N/C
L2CA_ConfigReq	CONFIG	Send peer L2CAP_ConfigReq message. Start RTX timer.	N/C

Table 3.1: L2CAP Channel State Machine

Event	Current State	Action	New State
L2CA_ConfigReq	OPEN	Suspend data transmission at a convenient point. Send peer L2CAP_ConfigReq message. Start RTX timer.	CONFIG
L2CA_ConfigRsp	CONFIG	Send peer L2CAP_ConfigRsp message. If all outstanding L2CAP_ConfigReq messages have received positive responses then move in OPEN state. Otherwise, remain in CONFIG state.	N/C or OPEN
L2CA_ConfigRspNeg	CONFIG	Send peer L2CAP_ConfigRspNeg message.	N/C
L2CA_Disconnect Req	OPEN or CONFIG	Send peer L2CAP_DisconnectReq message. Start RTX timer.	W4_L2CAP_ DISCONNECT _RSP
L2CA_DisconnectRsp	W4_L2CA_ DISCONNECT_ RSP	Send peer L2CAP_DisconnectRsp message. Return CID to free pool.	CLOSED
L2CA_DataRead	OPEN	If payload complete, transfer payload to InBuffer.	OPEN
L2CA_DataWrite	OPEN	Send peer L2CAP_Data message.	OPEN
Timer_RTX	Any	Send upper layer L2CA_TimeOutInd message. If final expiration, return CID to free pool else re-send Request.	CLOSED
Timer_ERTX	Any	Send upper layer L2CA_TimeOutInd message. If final expiration, return CID to free pool else re-send Request.	CLOSED

Table 3.1: L2CAP Channel State Machine

Figure 3.3 illustrates a simplified state machine and typical transition path taken by an initiator and acceptor. The state machine shows what events cause state transitions and what actions are also taken while the transitions occur. Not all the events listed in Table 3.1 are included in the simplified State Machine to avoid cluttering the figure.

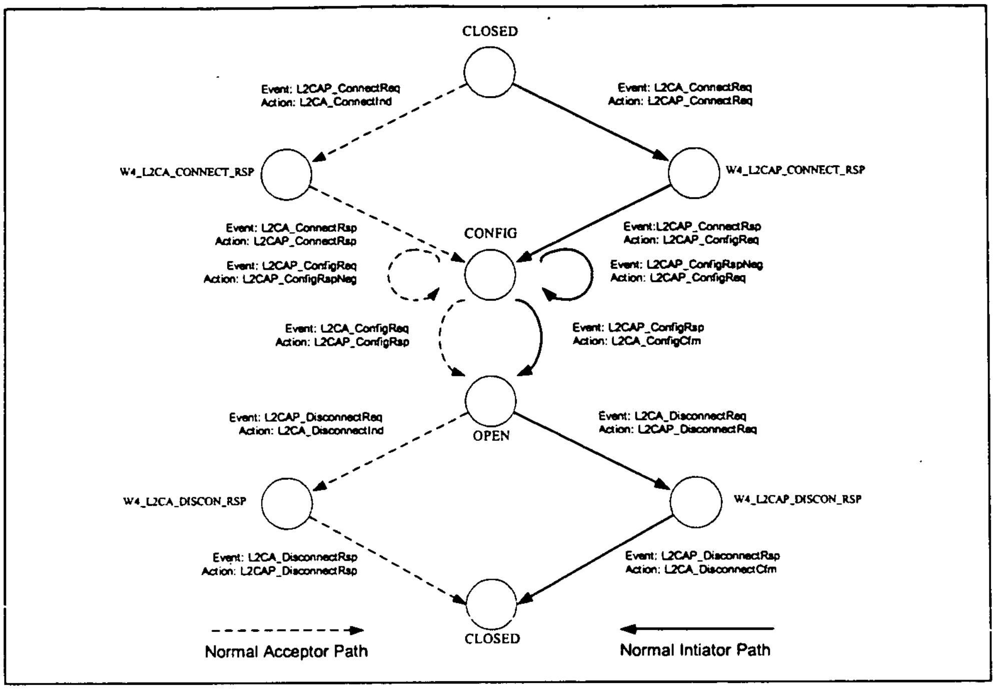


Figure 3.3: State Machine Example

Figure 3.4 presents another illustration of the events and actions based around the messages sequences being communicated between two devices. In this example, the initiator is creating the first L2CAP channel between two devices. Both sides start in the CLOSED state. After receiving the request from the upper layer, the entity requests the lower layer to establish a physical link. If no physical link exists, LMP commands are used to create the physical link between the devices. Once the physical link is established, L2CAP signals may be sent over it.

Figure 3.4 is an example and not all setup sequences will be identical to the one illustrated below.

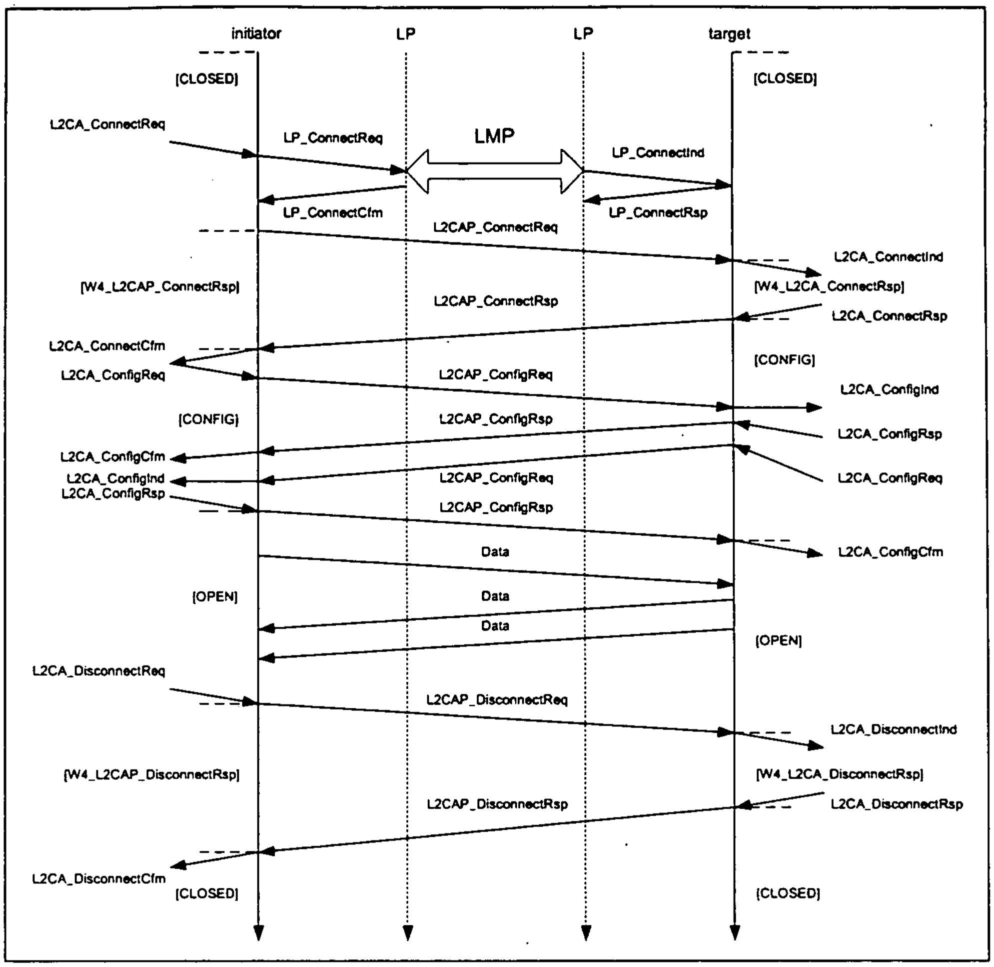


Figure 3.4: Message Sequence Chart of Basic Operation

4 DATA PACKET FORMAT

L2CAP is packet-based but follows a communication model based on *channels.* A channel represents a data flow between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless.
All packet fields use Little Endian byte order.

4.1 CONNECTION-ORIENTED CHANNEL

Figure 4.1 illustrates the format of the L2CAP packet (also referred to as the L2CAP PDU) within a connection-oriented channel.

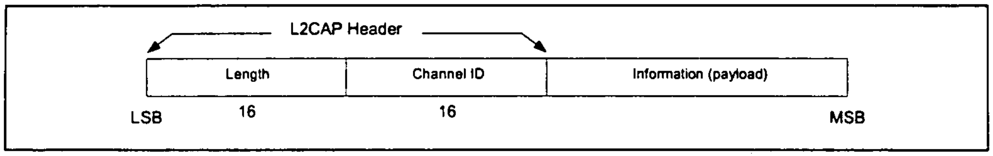


Figure 4.1: L2CAP Packet (field sizes in bits)

The fields shown are:

- *Length: 2 octets (16 bits)*

Length indicates the size of information payload in bytes, excluding the length of the L2CAP header. The length of an information payload can be up to 65535 bytes. The Length field serves as a simple integrity check of the reassembled L2CAP packet on the receiving end.

- *Channel ID: 2 octets*

  The channel ID identifies the destination channel endpoint of the packet. The scope of the channel ID is relative to the device the packet is being sent to.

- *Information: 0 to 65535 octets*

  This contains the payload received from the upper layer protocol (outgoing packet), or delivered to the upper layer protocol (incoming packet). The minimum supported MTU for connection-oriented packets (MTU_{cno}) is negotiated during channel configuration (see Section 6.1 on page 289). The minimum supported MTU for the signalling packet (MTU_{sig}) is 48 bytes (see Section 5 on page 275).

4.2 CONNECTIONLESS DATA CHANNEL

In addition to connection-oriented channels, L2CAP also exports the concept of a group-oriented channel. Data sent to the 'group' channel is sent to all members of the group in a best-effort manner. Groups have no quality of service associated with them. Group channels are unreliable; L2CAP makes no guarantee that data sent to the group successfully reaches all members of the group. If reliable group transmission is required, it must be implemented at a higher layer.

Transmissions to a group must be non-exclusively sent to all members of that group. The local device cannot be a member of the group, and higher layer protocols are expected to loopback any data traffic being sent to the local device. Non-exclusive implies non-group members may receive group transmissions and higher level (or link level) encryption can be used to support private communication.

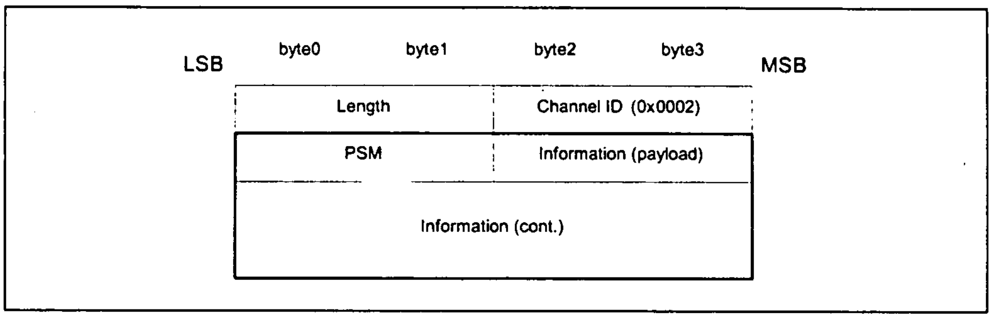


Figure 4.2: Connectionless Packet

The fields shown are:

- *Length: 2 octets*

  Length indicates the size of information payload plus the PSM field in bytes, excluding the length of the L2CAP header.

- *Channel ID: 2 octets*

  Channel ID (0x0002) reserved for connectionless traffic.

- *Protocol/Service Multiplexer (PSM): 2 octets (minimum)*

  The PSM field is based on the ISO 3309 extension mechanism for address fields. All content of the PSM field, referred to as the PSM value, must be ODD, that is, the least significant bit of the least significant octet must be '1'. Also, all PSM values must be assigned such that the least significant bit of the most significant octet equals '0'. This allows the PSM field to be extended beyond 16 bits. The PSM value definitions are specific to L2CAP and assigned by the Bluetooth SIG. For more information on the PSM field see Section 5.2 on page 278.

- *Information: 0 to 65533 octets*

  The payload information to be distributed to all members of the group. Implementations must support a minimum connectionless MTU (MTU_{cnl}) of 670 octets, unless explicitly agreed upon otherwise, e.g., for single operation devices that are built to comply to a specific Bluetooth profile that dictates the use of a specific MTU for connectionless traffic that is less than MTU_{cnl}.

The L2CAP group service interface provides basic group management mechanisms including creating a group, adding members to a group, and removing members from a group. There are no pre-defined groups such as 'all radios in range'.

5 SIGNALLING

This section describes the signalling commands passed between two L2CAP entities on remote devices. All signalling commands are sent to CID 0x0001. The L2CAP implementation must be able to determine the Bluetooth address (BD_ADDR) of the device that sent the commands. Figure 5.1 illustrates the general format of all L2CAP packets containing signalling commands. Multiple commands may be sent in a single (L2CAP) packet and packets are sent to CID 0x0001. MTU Commands take the form of Requests and Responses. All L2CAP implementations must support the reception of signalling packets whose MTU (MTU_{sig}) does not exceed 48 bytes. L2CAP implementations should not use signalling packets beyond this size without first testing whether the implementation can support larger signalling packets.

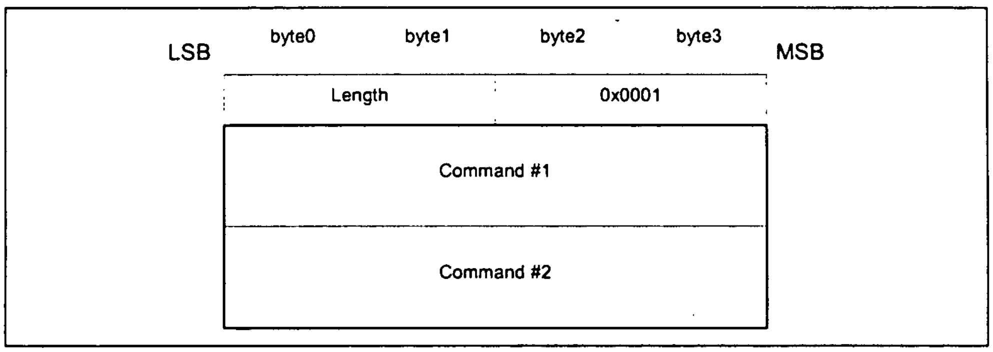


Figure 5.1: Signalling Command Packet Format

Figure 5.2 displays the general format of all signalling commands.

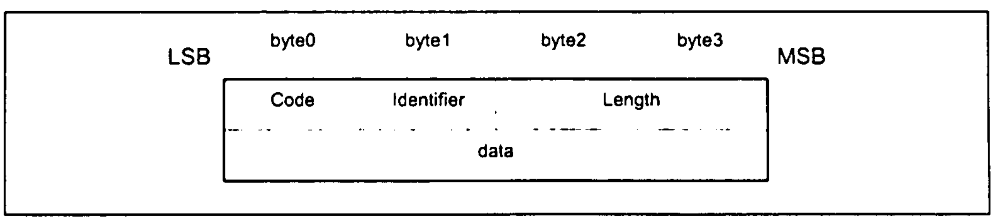


Figure 5.2: Command format

The fields shown are:

- *Code: 1 octet*

  The Code field is one octet long and identifies the type of command. When a packet is received with an unknown Code field, a Command Reject packet (defined in Section 5.1 on page 277) is sent in response.

  Up-to-date values of assigned Codes are specified in the latest Bluetooth 'Assigned Numbers' document (page 1009). Table 5.1 on page 276 lists the codes defined by this document. All codes are specified with the most significant bit in the left-most position.

Code	Description
0x00	RESERVED
0x01	Command reject
0x02	Connection request
0x03	Connection response
0x04	Configure request
0x05	Configure response
0x06	Disconnection request
0x07	Disconnection response
0x08	Echo request
0x09	Echo response
0x0A	Information request
0x0B	Information response

Table 5.1: Signalling Command Codes

- *Identifier: 1 octet*

  The Identifier field is one octet long and helps matching a request with the reply. The requesting device sets this field and the responding device uses the same value in its response. A different Identifier must be used for each original command. Identifiers should not be recycled until a period of 360 seconds has elapsed from the initial transmission of the command using the identifier. On the expiration of a RTX or ERTX timer, the same identifier should be used if a duplicate Request is re-sent as stated in Section 3.1.5 on page 262. A device receiving a duplicate request should reply with a duplicate response. A command response with an invalid identifier is silently discarded. Signalling identifier 0x0000 is defined to be an illegal identifier and shall never be used in any command.

- *Length: 2 octets*

  The Length field is two octets long and indicates the size in octets of the data field of the command only, i.e., it does not cover the Code, Identifier, and Length fields.

- *Data: 0 or more octets*

  The Data field is variable in length and discovered using the Length field. The Code field determines the format of the Data field.

5.1 COMMAND REJECT (CODE 0x01)

A Command Reject packet is sent in response to a command packet with an unknown command code or when sending the corresponding Response is inappropriate. Figure 5.3 displays the format of the packet. The Identifier should match the Identifier of the packet containing the unidentified code field. Implementations must always send these packets in response to unidentified signalling packets.

When multiple commands are included in an L2CAP packet and the packet exceeds the MTU of the receiver, a single Command Reject packet is sent in response. The identifier should match the first Request command in the L2CAP packet. If only Responses are recognized, the packet shall be silently discarded.

LSB byte0 byte1 byte2 byte3 MSB

Code=0x01	Identifier	Length	
Reason		Data (optional)	

Figure 5.3: Command Reject Packet

- *Length = 0x0002 or more octets*
- *Reason: 2 octets*

  The Reason field describes why the Request packet was rejected.

Reason value	**Description**
0x0000	Command not understood
0x0001	Signalling MTU exceeded
0x0002	Invalid CID in request
Other	Reserved

Table 5.2: Reason Code Descriptions

- *Data: 0 or more octets*

  The length and content of the Data field depends on the Reason code. If the Reason code is 0x0000, “Command not understood”, no Data field is used. If the Reason code is 0x0001, “Signalling MTU Exceeded”, the 2-octet Data field represents the maximum signalling MTU the sender of this packet can accept.

  If a command refers to an invalid channel then the Reason code 0x0002 will be returned. Typically a channel is invalid because it does not exist. A 4-octet data field on the command reject will contain the local (first) and remote (second) channel endpoints (relative to the sender of the Command Reject) of the disputed channel. The latter endpoints are obtained from the corresponding rejected command. If the rejected command contains only one of the channel endpoints, the other one is replaced by the null CID 0x0000.

Reason value	Data Length	Data value
0x0000	0 octets	N/A
0x0001	2 octets	Actual MTU
0x0002	4 octets	Requested CID

Table 5.3: Reason Data values

5.2 CONNECTION REQUEST (CODE 0x02)

Connection request packets are sent to create a channel between two devices. The channel connection must be established before configuration may begin. Figure 5.4 illustrates a Connection Request packet.

LSB byte0 byte1 byte2 byte3 MSB

byte0	byte1	byte2	byte3
Code=0x02	Identifier	Length	
PSM		Source CID	

Figure 5.4: Connection Request Packet

- *Length = 0x0004 or more octets*
- *Protocol/Service Multiplexor (PSM): 2 octets (minimum)*

  The PSM field is two octets (minimum) in length. The structure of the PSM field is based on the ISO 3309 extension mechanism for address fields. All PSM values must be ODD, that is, the least significant bit of the least significant octet must be '1'. Also, all PSM values must be assigned such that the least significant bit of the most significant octet equals '0'. This allows the PSM field to be extended beyond 16 bits. PSM values are separated into two ranges. Values in the first range are assigned by the Bluetooth SIG and indicate protocols. The second range of values are dynamically allocated and used in conjunction with the Service Discovery Protocol (SDP). The dynamically assigned values may be used to support multiple implementations of a particular protocol, e.g., RFCOMM, residing on top of L2CAP or for prototyping an experimental protocol.

PSM value	Description
0x0001	Service Discovery Protocol
0x0003	RFCOMM
0x0005	Telephony Control Protocol
<0x1000	RESERVED
[0x1001-0xFFFF]	DYNAMICALLY ASSIGNED

Table 5.4: Defined PSM Values

- *Source CID (SCID): 2 octets*

  The source local CID is two octets in length and represents a channel end-point on the device sending the request. Once the channel has been configured, data packets flowing from the sender of the request must be send to this CID. In this section, the Source CID represents the channel endpoint on the device sending the request and receiving the response, while the Destination CID represents the channel endpoint on the device receiving the request and sending the response.

5.3 CONNECTION RESPONSE (CODE 0x03)

When a unit receives a Connection Request packet, it must send a Connection Response packet. The format of the connection response packet is shown in Figure 5.5.

LSB	byte0	byte1	byte2	byte3	MSB
	Code=0x03	Identifier	Length		
	Destination CID		Source CID		
	Result		Status		

Figure 5.5: Connection Response Packet

- *Length = 0x0008 octets*
- *Destination Channel Identifier (DCID): 2 octets*

  The field contains the channel end-point on the device sending this Response packet.
- *Source Channel Identifier (SCID): 2 octets*

  The field contains the channel end-point on the device receiving this Response packet.

- *Result: 2 octets*

  The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a non-zero value indicates the connection request failed. A logical channel is established on the receipt of a successful result. Table 5.5 defines values for this field. If the result field is not zero, the DCID and SCID fields should be ignored.

Value	Description
0x0000	Connection successful.
0x0001	Connection pending
0x0002	Connection refused – PSM not supported.
0x0003	Connection refused – security block.
0x0004	Connection refused – no resources available.
Other	Reserved.

Table 5.5: Result values

- *Status: 2 octets*

  Only defined for Result = Pending. Indicates the status of the connection.

Value	Description
0x0000	No further information available
0x0001	Authentication pending
0x0002	Authorization pending
Other	Reserved

Table 5.6: Status values

5.4 CONFIGURATION REQUEST (CODE 0x04)

Configuration Request packets are sent to establish an initial logical link transmission contract between two L2CAP entities and also to re-negotiate this contract whenever appropriate. During a re-negotiation session, all data traffic on the channel should be suspended pending the outcome of the negotiation. Each configuration parameter in a Configuration Request is related exclusively either with the outgoing or the incoming data traffic but not both of them. In Section 6 on page 289, the various configuration parameters and their relation to the outgoing or incoming data traffic are presented. If an L2CAP entity receives a Configuration Request while it is waiting for a response it must not block sending the Configuration Response, otherwise the configuration process may deadlock.

If no parameters need to be negotiated, no options need to be inserted and the C-bit should be cleared. L2CAP entities in remote devices MUST negotiate all parameters defined in this document whenever the default values are not

acceptable. Any missing configuration parameters are assumed to have their most recently (mutually) explicitly or implicitly accepted values. Event if all default values are acceptable, a Configuration Request packet with no options MUST be sent. Implicitly accepted values are any default values for the configuration parameters specified in this document that have not been explicitly negotiated for the specific channel under configuration.

Each configuration parameter is one-directional and relative to the direction implied by the sender of a Configuration Request. If a device needs to establish the value of a configuration parameter in the opposite direction than the one implied by a Configuration Request, a new Configuration Request with the desired value of the configuration parameter in it needs to be sent in the direction opposite the one used for the original Connection Request.

The decision on the amount of time (or messages) spent arbitrating the channel parameters before terminating the negotiation is left to the implementation but it shall not last more than 120 seconds.

Figure 5.6 defines the format of the Configuration Request packet.

LSB byte0	byte1	byte2	byte3 MSB
Code=0x04	Identifier	Length	
Destination CID		Flags	
Options			

Figure 5.6: Configuration Request Packet

- *Length = 0x0004 or more octets*
- *Destination CID (DCID): 2 octets*

  The field contains the channel end-point on the device receiving this Request packet.
- *Flags: 2 octets*

  Figure 5.7 display the two-octet Flags field. Note the most significant bit is shown on the left.

MSB Reserved	C LSB

Figure 5.7: Configuration Request Flags field format

C – more configuration requests will follow when set to 1. This flag indicates that the remote device should not enter OPEN state after agreeing to these parameters because more parameter negotiations are being sent. Segment-

ing the Configuration Request packet is necessary if the parameters exceed the MTU_{sig}.

Other flags are reserved and should be cleared. L2CAP implementations should ignore these bits.

- *Configuration Options*

  The list of the parameters and their values to be negotiated. These are defined in Section 6 on page 289. Configuration Requests may contain no options (referred to as an empty or null configuration request) and can be used to request a response. For an empty configuration request the length field is set to 0x0004.

5.5 CONFIGURE RESPONSE (CODE 0X05)

Configure Response packets MUST be sent in reply to Configuration Request packets. Each configuration parameter value (if any is present) in a Configuration Response reflects an 'adjustment' to a configuration parameter value that has been sent (or, in case of default values, implied) in the corresponding Configuration Request. Thus, for example, if a configuration parameter in a Configuration Request relates to traffic flowing from device A to device B, the sender of the Configuration Response will only adjust (if needed) this value again for the same traffic flowing from device A to device B. The options sent in the Response depend on the value in the Result field. Figure 5.8 defines the format of the Configuration Response packet.

LSB	byte0	byte1	byte2	byte3	MSB
	Code=0x05	Identifier	Length		
	Source CID		Flags		
	Result		Config		

Figure 5.8: Configuration Response Packet

- *Length = 0x0006 or more octets*
- *Source CID (SCID): 2 octets*

  The field contains the channel end-point on the device receiving this Response packet. The device receiving the Response must check that the Identifier field matches the same field in the corresponding configuration request command and the SCID matches its local CID paired with the original DCID.
- *Flags: 2 octets*

  Figure 5.9 displays the two-octet Flags field. Note the most significant bit is shown on the left.

MSB	LSB
Reserved	C

Figure 5.9: Configuration Response Flags field format

C – more configuration responses will follow when set to 1. This flag indicates that the parameters included in the response are a partial subset of parameters being sent by the device sending the Response packet.

Other flags are reserved and should be cleared. L2CAP implementations should ignore these bits.

- *Result: 2 octets*

  The Result field indicates whether or not the Request was acceptable. See Table 5.7 for possible result codes.

Result	Description
0x0000	Success
0x0001	Failure – unacceptable parameters
0x0002	Failure – rejected (no reason provided)
0x0003	Failure – unknown options
Other	RESERVED

Table 5.7: Configuration Response Result codes

- *Configuration Options*

  This field contains the list of parameters being negotiated. These are defined in Section 6 on page 289. On a successful result, these parameters contain the return values for any wild card parameters (see Section 6.3 on page 291) contained in the request.

  On an unacceptable parameters failure (Result=0x0001) the rejected parameters should be sent in the response with the values that would have been accepted if sent in the original request. Any missing configuration parameters are assumed to have their most recently (mutually) accepted values and they too can be included in the Configuration Response if need to be changed. Recall that, each configuration parameter is one-directional and relative to the direction implied by the sender of a Configuration Request. Thus, if the sender of the Configuration Response needs to establish the value of a configuration parameter in the opposite direction than the one implied by an original Configuration Request, a new Configuration Request with the desired value of the configuration parameter in it needs to be sent in the direction opposite the one used for the original Connection Request.

  On an unknown option failure (Result=0x0003), the option types not understood by the recipient of the Request must be included in the Response. Note that hints (defined in Section 6 on page 289), those options in the Request that are skipped if not understood, must not be included in the Response and must not be the sole cause for rejecting the Request.

  The decision on the amount of time (or messages) spent arbitrating the channel parameters before terminating the negotiation is left to the implementation.

5.6 DISCONNECTION REQUEST (CODE 0x06)

Terminating an L2CAP channel requires that a disconnection request packet be sent and acknowledged by a disconnection response packet. Disconnection is requested using the signalling channel since all other L2CAP packets sent to the destination channel automatically get passed up to the next protocol layer. Figure 5.10 displays a disconnection packet request. The receiver must ensure both source and destination CIDs match before initiating a connection disconnection. Once a Disconnection Request is issued, all incoming data in transit on this L2CAP channel will be discarded and any new additional outgoing data is not allowed. Once a disconnection request for a channel has been received, all data queued to be sent out on that channel may be discarded.

LSB	byte0	byte1	byte2	byte3	MSB
	Code=0x06	Identifier	Length		
	Destination CID		Source CID		

Figure 5.10: Disconnection Request Packet

- *Length = 0x0004 octets*
- *Destination CID (DCID): 2 octets*

  This field specifies the end-point of the channel to be shutdown on the device receiving this request.
- *Source CID (SCID): 2 octets*

  This field specifies the end-point of the channel to be shutdown on the device sending this request.

  The SCID and DCID are relative to the sender of this request and must match those of the channel to be disconnected. If the DCID is not recognized by the receiver of this message, a CommandReject message with 'invalid CID' result code must be sent in response. If the receivers finds a DCID match but the SCID fails to find the same match, the request should be silently discarded.

5.7 DISCONNECTION RESPONSE (CODE 0x07)

Disconnection responses should be sent in response to each disconnection request.

LSB	byte0	byte1	byte2	byte3	MSB
	Code=0x07	Identifier	Length		
	Destination CID		Source CID		

Figure 5.11: Disconnection Response Packet

- *Length = 0x0004 octets*
- *Destination CID (DCID): 2 octets*

  This field identifies the channel end-point on the device sending the response.
- *Source CID (SCID): 2 octets*

  This field identifies the channel end-point on the device receiving the response.

  The DCID and the SCID (which are relative to the sender of the request), and the Identifier fields must match those of the corresponding disconnection request command. If the CIDs do not match, the response should be silently discarded at the receiver.

5.8 ECHO REQUEST (CODE 0x08)

Echo requests are used to solicit a response from a remote L2CAP entity. These requests may be used for testing the link or passing vendor specific information using the optional data field. L2CAP entities MUST respond to well-formed Echo Request packets with an Echo Response packet. The Data field is optional and implementation-dependent. L2CAP entities should ignore the contents of this field.

LSB	byte0	byte1	byte2	byte3	MSB
	Code=0x08	Identifier	Length		
	Data (optional)				

Figure 5.12: Echo Request Packet

5.9 ECHO RESPONSE (CODE 0x09)

Echo responses are sent upon receiving Echo Request packets. The identifier in the response MUST match the identifier sent in the Request. The optional and implementation-dependent data field may contain the contents of the data field in the Request, different data, or no data at all.

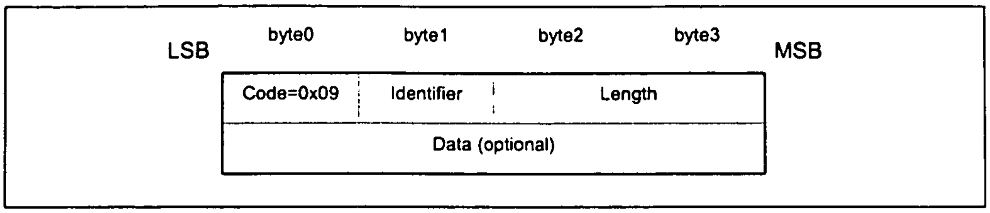


Figure 5.13: Echo Response Packet

5.10 INFORMATION REQUEST (CODE 0X0A)

Information requests are used to solicit implementation-specific information from a remote L2CAP entity. L2CAP entities MUST respond to well-formed Information Request packets with an Information Response packet.

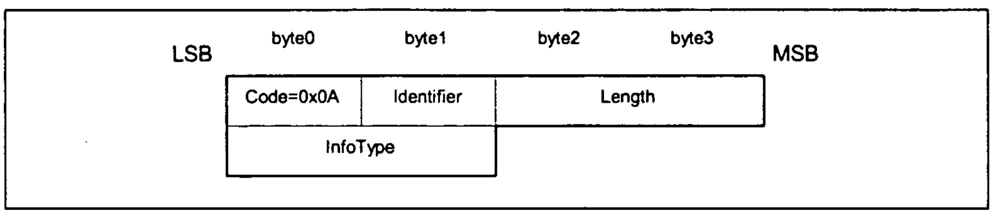


Figure 5.14: Information Request Packet

- *Length = 0x0002 octets*
- *InfoType: 2 octets*
- The InfoType defines the type of implementation-specific information being solicited.

Value	**Description**
0x0001	Connectionless MTU
Other	Reserved

Table 5.8: InfoType definitions

5.11 INFORMATION RESPONSE (CODE 0X0B)

Information responses are sent upon receiving Information Request packets. The identifier in the response MUST match the identifier sent in the Request. The optional data field may contain the contents of the data field in the Request, different data, or no data at all.

LSB byte0 byte1 byte2 byte3 MSB

Code=0x0B	Identifier	Length	
InfoType		Result	
Data (optional)			

Figure 5.15: Information Response Packet

- *InfoType: 2 octets*

  Same value sent in the request.

- *Result: 2 octets*

  The Result contains information about the success of the request. If result is "Success", the data field contains the information as specified in Table 5.10. If result is "Not supported", no data should be returned.

Value	**Description**
0x0000	Success
0x0001	Not supported
Other	Reserved

Table 5.9: Information Response Result values

- *Data: 0 or more octets*

  The contents of the Data field depends on the InfoType. For the Connection MTU request, the data field contains the remote entity's 2-octet acceptable connectionless MTU.

InfoType	**Data**	**Data Length (in octets)**
0x0001	Connectionless MTU	2

Table 5.10: Information Response Data fields

6 CONFIGURATION PARAMETER OPTIONS

Options are a mechanism to extend the ability to negotiate different connection requirements. Options are transmitted in the form of information elements comprised an option type, an option length, and one or more option data fields. Figure 6.1 illustrates the format of an option.

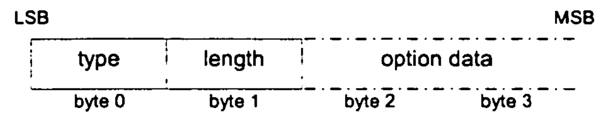


Figure 6.1: Configuration option format

- *Type: 1 octet*

  The option type field defines the parameters being configured. The most significant bit of the type determines the action taken if the option is not recognized. The semantics assigned to the bit are defined below.

  0 - option must be recognized; refuse the configuration request

  1 - option is a hint; skip the option and continue processing

- *Length: 1 octet*

  The length field defines the number of octets in the option payload. So an option type with no payload has a length of 0.

- *Option data*

  The contents of this field are dependent on the option type.

6.1 MAXIMUM TRANSMISSION UNIT (MTU)

This option specifies the payload size the sender is capable of accepting. The type is 0x01, and the payload length is 2 bytes, carrying the two-octet MTU size value as the only information element (see Figure 6.2 on page 290).

Since all L2CAP implementations are capable to support a minimum L2CAP packet size, see Section 4 on page 272, MTU is not really a negotiated value but rather an informational parameter to the remote device that the local device can accommodate in this channel an MTU larger than the minimum required. In the unlikely case that the remote device is only willing to send L2CAP packets in this channel that are larger than the MTU announced by the local device, then this Configuration Request will receive a negative response in which the remote device will include the value of MTU that is indented to transmit. In this case, it is implementation specific on whether the local device will continue the configuration process or even maintain this channel.

The remote device in its positive Configuration Response will include the actual MTU to be used on this channel for traffic flowing into the local device which is

minimum{ MTU in configReq, outgoing MTU capability of remote device }. The MTU to be used on this channel but for the traffic flowing in the opposite direction will be established when the remote device (with respect to this discussion) sends its own Configuration Request as explained in Section 5.4 on page 280.

0 31

type=0x01	length=2	MTU

Figure 6.2: MTU Option Format

- Maximum Transmission Unit (MTU) Size: 2 octets

  The MTU field represents the largest L2CAP packet payload, in bytes, that the originator of the Request can accept for that channel. The MTU is asymmetric and the sender of the Request shall specify the MTU it can receive on this channel if it differs from the default value. L2CAP implementations must support a minimum MTU size of 48 bytes. The default value is 672 bytes[1].

6.2 FLUSH TIMEOUT OPTION

This option is used to inform the recipient of the amount of time the originator's link controller / link manager will attempt to successfully transmit an L2CAP segment before giving up and flushing the packet. The type is 0x02 and the payload size is 2 octets.

0 31

type=0x02	length=2	Flush Timeout

Figure 6.3: Flush Timeout

- *Flush Timeout*

  This value represents units of time measured in milliseconds. The value of 1 implies no retransmissions at the Baseband level should be performed since the minimum polling interval is 1.25 ms. The value of all 1's indicates an infinite amount of retransmissions. This is also referred to as 'reliable channel'. In this case, the link manager shall continue retransmitting a segment until physical link loss occurs. This is an asymmetric value and the sender of the Request shall specify its flush timeout value if it differs from the default value of 0xFFFF.

1. The default MTU was selected based on the payload carried by two Baseband DH5 packets (2*341=682) minus the Baseband ACL headers (2*2=4) and L2CAP header (6).

6.3 QUALITY OF SERVICE (QOS) OPTION

This option specifies a flow specification (flowSpec) similar to RFC 1363 [1]. If no QoS configuration parameter is negotiated the link should assume the default parameters discussed below. The QoS option is type 0x03.

When included in a Configuration Request, this option describes the outgoing traffic flow from the device sending the request to the device receiving it. When included in a positive Configuration Response, this option describes the incoming traffic flow agreement as seen from the device sending the response. When included in a negative Configuration Response, this option describes the preferred incoming traffic flow from the perspective of the device sending the response.

L2CAP implementations are only required to support 'Best Effort' service, support for any other service type is optional. Best Effort does not require any guarantees. If no QoS option is placed in the request, Best Effort must be assumed. If any QoS guarantees are required then a QoS configuration request must be sent.

The remote device places information that depends on the value of the result field, see Section 5.5 on page 283, in its Configuration Response. If the request was for Guaranteed Service, the response shall include specific values for any wild card parameters (see Token Rate and Token Bucket Size descriptions) contained in the request. If the result is "Failure – unacceptable parameters", the response may include a list of outgoing flowspec parameters and parameter values that would make a new Connection Request from the local device acceptable by the remote device. Both explicitly referenced in a Configuration Request or implied configuration parameters can be included in a Configuration Response. Recall that any missing configuration parameters from a Configuration Request are assumed to have their most recently (mutually) accepted values. For both Best effort and Guaranteed service, when the QoS option appears in the Configuration Response, "do not cares" shall be present where they appeared in the Configuration Request.

0			31
0x03	length=22	Flags	service type
Token Rate			
Token Bucket Size (bytes)			
Peak Bandwidth (bytes/second)			
Latency (microseconds)			
Delay Variation (microseconds)			

Figure 6.4: Quality of Service Flow Specification

- *Flags: 1 octet*

  Reserved for future use and must be set to 0.

- *Service Type: 1 octet*

  This field indicates the level of service required. Table 6.1 defines the different services available. If 'No traffic' is selected, the remainder of the fields may be ignored because there is no data being sent across the channel in the outgoing direction.

  If 'Best effort', the default value, is selected, the remaining fields should be treated as hints by the remote device. The remote device may choose to ignore the fields, try to satisfy the hint but provide no response (QoS option omitted in the Response message), or respond with the settings it will try to meet.

Value	Description
0x00	No traffic
0x01	Best effort (Default)
0x02	Guaranteed
Other	Reserved

Table 6.1: Service type definitions

- *Token Rate: 4 octets*

  The value of this field represents the rate at which traffic credits are granted in bytes per second. An application may send data at this rate continuously. Burst data may be sent up to the token bucket size (see below). Until that data burst has been drained, an application must limit itself to the token rate. The value 0x00000000 indicates no token rate is specified. This is the default value and implies indifference to token rate. The value 0xFFFFFFFF represents a wild card matching the maximum token rate available. The meaning of this value depends on the semantics associated with the service type. For best effort, the value is a hint that the application wants as much bandwidth as possible. For Guaranteed service the value represents the maximum bandwidth available at the time of the request.

- *Token Bucket Size: 4 octets*

  The value of this field represents the size of the token bucket in bytes. If the bucket is full, then applications must either wait or discard data. The value of 0x00000000 represents no token bucket is needed; this is the default value. The value 0xFFFFFFFF represents a wild card matching the maximum token bucket available. The meaning of this value depends on the semantics associated with the service type. For best effort, the value indicates the application wants a bucket as big as possible. For Guaranteed service the value represents the maximum buffer space available at the time of the request.

- *Peak Bandwidth: 4 octets*

The value of this field, expressed in bytes per second, limits how fast packets may be sent back-to-back from applications. Some intermediate systems can take advantage of this information, resulting in more efficient resource allocation. The value of 0x00000000 states that the maximum bandwidth is unknown, which is the default value.

- *Latency: 4 octets*

  The value of this field represents the maximum acceptable delay between transmission of a bit by the sender and its initial transmission over the air, expressed in microseconds. The precise interpretation of this number depends on the level of guarantee specified in the Class of Service. The value 0xFFFFFFFF represents a do not care and is the default value.

- *Delay Variation: 4 octets*

  The value of this field is the difference, in microseconds, between the maximum and minimum possible delay that a packet will experience. This value is used by applications to determine the amount of buffer space needed at the receiving side in order to restore the original data transmission pattern. The value 0xFFFFFFFF represents a do not care and is the default value.

6.4 CONFIGURATION PROCESS

Negotiating the channel parameters involves three steps:

1. Informing the remote side of the non-default parameters that the local side will accept
2. Having the remote side agreeing or disagreeing to these values (including the default ones); steps (1) and (2) may iterate as needed
3. Repeat steps (1) and (2) for the reverse direction from the (previous) remote side to the (previous) local side.

This process can be abstracted into a Request negotiation path and a Response negotiation path.

6.4.1 Request Path

The Request Path negotiates the incoming MTU, flush timeout, and outgoing flowspec. Table 6.2 defines the configuration options that may be placed in the Configuration Request message and their semantics.

Parameter	Description
MTU	Incoming MTU information
FlushTO	Outgoing flush timeout
OutFlow	Outgoing flow information.

Table 6.2: Parameters allowed in Request

6.4.2 Response Path

The Response Path negotiates the outgoing MTU (remote side's incoming MTU), the remote side's flush timeout, and incoming flowspec (remote side's outgoing flowspec). If a request-oriented parameter is not present in the Request message (reverts to default value), the remote side may negotiate for a non-default value by including the proposed value in a negative Response message.

Parameter	Description
MTU	Outgoing MTU information
FlushTO	Incoming flush timeout
InFlow	Incoming flow information

Table 6.3: Parameters allowed in Response

6.4.3 Configuration State Machine

The configuration state machine shown below depicts two paths. Before leaving the CONFIG state and moving into the OPEN state, both paths must reach closure. The request path requires the local device to receive a positive response to reach closure while the response path requires the local device to send a positive response to reach closure.

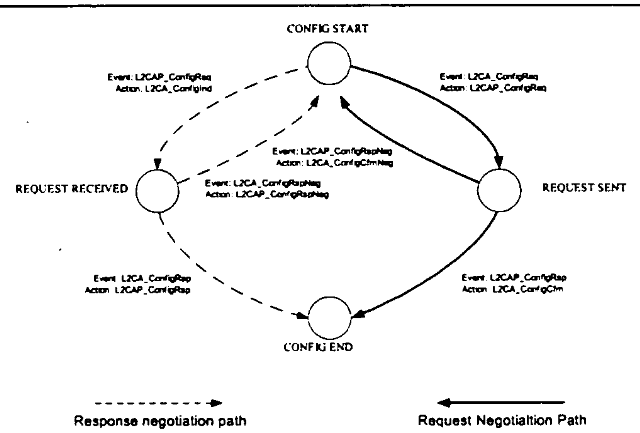


Figure 6.5: Configuration State Machine

"Appendix A: Configuration MSCs" on page 318 provides some configuration examples.

7 SERVICE PRIMITIVES

This section presents an abstract description of the services offered by L2CAP in terms of service primitives and parameters. The service interface is required for testing. The interface is described independently of any platform specific implementation. All data values use Little Endian byte ordering.

7.1 EVENT INDICATION

Service	Input Parameters	Output Parameters
EventIndication	Event, Callback	Result

Description:

The use of this primitive requests a callback when the selected indication Event occurs.

Input Parameters:

Event *Type: uint* *Size: 2 octets*

Value	Description
0x00	Reserved
0x01	L2CA_ConnectInd
0x02	L2CA_ConfigInd
0x03	L2CA_DisconnectInd
0x04	L2CA_QoSViolationInd
other	Reserved for future use

Callback *Type: function* *Size: N/A*

Event	Callback Function Input Parameters
L2CA_ConnectInd	BD_ADDR, CID, PSM, Identifier
L2CA_ConfigInd	CID, OutMTU, InFlow, InFlushTO
L2CA_DisconnectInd	CID
L2CA_QoSViolationInd	BD_ADDR

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Event successfully registered
0x0001	Event registration failed

7.1.1 L2CA_ConnectInd Callback

This callback function includes the parameters for the address of the remote device that issued the connection request, the local CID representing the channel being requested, the Identifier contained in the request, and the PSM value the request is targeting.

7.1.2 L2CA_ConfigInd Callback

This callback function includes the parameters indicating the local CID of the channel the request has been sent to, the outgoing MTU size (maximum packet that can be sent across the channel) and the flowspec describing the characteristics of the incoming data. All other channel parameters are set to their default values if not provided by the remote device.

7.1.3 L2CA_DisconnectInd Callback

This callback function includes the parameter indicating the local CID the request has been sent to.

7.1.4 L2CA_QoSViolationInd Callback

This callback function includes the parameter indicating the address of the remote Bluetooth device where the QoS contract has been violated.

7.2 CONNECT

Service	**Input Parameters**	**Output Parameters**
L2CA_ConnectReq	PSM, BD_ADDR	LCID, Result, Status

Description:

This primitive initiates the sending of an L2CA_ConnectReq message and blocks until a corresponding L2CA_ConnectCfm(Neg) or L2CA_TimeOutInd message is received.

The use of this primitive requests the creation of a channel representing a logical connection to a physical address. Input parameters are the target protocol (*PSM*) and remote device's 48-bit address (*BD_ADDR*). Output parameters are the local CID *(LCID)* allocated by the local L2CAP entity, and *Result* of the request. If the *Result* indicates success, the *LCID* value contains the identification of the local endpoint. Otherwise the *LCID* returned should be set to 0. If *Result* indicates a pending notification, the *Status* value may contain more information of what processing is delaying the establishment of the connection. Otherwise the *Status* value should be ignored.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Target PSM provided for the connection

BD_ADDR *Type: unit* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device

Output Parameters:

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel if Result = 0x0000, otherwise set to 0.

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Connection successful and the CID identifies the local endpoint. Ignore Status parameter
0x0001	Connection pending. Check Status parameter for more information
0x0002	Connection refused because no service for the PSM has been registered
0x0003	Connection refused because the security architecture on the remote side has denied the request
0xEEEE	Connection timeout occurred. This is a result of a timer expiration indication being included in the connection confirm message

Status *Type: uint* *Size: 2 octets*

Value	Description
0x0000	No further information
0x0001	Authentication pending
0x0002	Authorization pending

7.3 CONNECT RESPONSE

Service	Input Paramet r	Output Parameters
L2CA_ConnectRsp	BD_ADDR, Identifier, LCID, Response, Status	Result

Description:

This primitive represents the L2CA_ConnectRsp.

The use of this primitive issues a response to a connection request event indication. Input parameters are the remote device's 48-bit address, Identifier sent in the request, local CID, the Response code, and the Status attached to the Response code. The output parameter is the Result of the service request.

This primitive must be called no more than once after receiving the callback indication. This primitive returns once the local L2CAP entity has validated the request. A successful return does indicate the response has been sent over the air interface.

Input Parameters:

BD_ADDR *Type: unit* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device

Identifier *Type: uint* *Size: 1 octets*

Value	Description
0xXX.	This value must match the value received in the L2CA_ConnectInd event described in Section 7.1.1 on page 296

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Response *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Connection successful
0x0001	Connection pending
0x0002	Connection refused – PSM not supported
0x0003	Connection refused – security block
0x0004	Connection refused – no resources available
0xXXXX	Other connection response code

Status *Type: uint* *Size: 2 octets*

Value	Description
0x0000	No further information available
0x0001	Authentication pending
0x0002	Authorization pending
0xXXXX	Other status code

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response successfully sent
0x0001	Failure to match any outstanding connection request

7.4 CONFIGURE

Service	Input Parameters	Output Parameters
L2CA_ConfigReq	CID, InMTU, OutFlow, OutFlushTO, LinkTO	Result, InMTU, OutFlow, OutFlushTO

Description:

This primitive initiates the sending of an L2CA_ConfigReq message and blocks until a corresponding L2CA_ConfigCfm(Neg) or L2CA_TimeOutInd message is received.

The use of this primitive requests the initial configuration (or reconfiguration) of a channel to a new set of channel parameters. Input parameters are the local CID endpoint, new incoming receivable MTU (InMTU), new outgoing flow specification, and flush and link timeouts. Output parameters composing the L2CA_ConfigCfm(Neg) message are the Result, accepted incoming MTU(InMTU), the remote side's flow requests, and flush and link timeouts. Note that the output results are returned only after the local L2CAP entity transitions out of the CONFIG state (even if this transition is back to the CONFIG state).

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Local CID

InMTU *Type: uint* *Size: 2 octets*

Valu	Description
0xXXXX	Maximum transmission unit this channel can accept

OutFlow *Type: Flow* *Size: x octets*

Value	Description
flowspec	Quality of service parameters dealing with the traffic characteristics of the outgoing data flow

OutFlushTO *Size 2 octets*

Value	Description
0xXXXX	Number of milliseconds to wait before an L2CAP packet that cannot be acknowledged at the physical layer is dropped
0x0000	Request to use the existing flush timeout value if one exists, otherwise the default value (0xFFFF) will be used
0x0001	Perform no retransmissions at the Baseband layer
0xFFFF	Perform retransmission at the Baseband layer until the link timeout terminates the channel

LinkTO *Size 2 octets*

Value	Description
0xXXXX	Number of milliseconds to wait before terminating an unresponsive link

Output Parameters:

Result *Size 2 octets*

Value	Description
0x0000	Configuration is successful. Parameters contain agreed upon values
0x0001	Failure – invalid CID
0x0002	Failure – unacceptable parameters
0x0003	Failure – signalling MTU exceeded
0x0004	Failure – unknown options
0xEEEE	Configuration timeout occurred. This is a result of a timer expiration indication being included in the configuration confirm

InMTU *Size 2 octets*

Value	Description
0xXXXX	Maximum transmission unit that the remote unit will send across this channel (maybe less or equal to the InMTU input parameter).

OutFlow *Size 2 octets*

Value	Description
FlowSpec	Quality of service parameters dealing with the traffic characteristics of the agreed-upon outgoing data flow if Result is successful. Otherwise this represents the requested Quality of Service

OutFlushTO *Size 2 octets*

Value	Description
0xXXXX	Number of milliseconds before an L2CAP packet that cannot be acknowledged at the physical layer is dropped. This value is informative of the actual value that will be used for outgoing packets. It may be less or equal to the OutFlushTO parameter given as input.

7.5 CONFIGURATION RESPONSE

Service	Input Parameters	Output Parameters
L2CA_ConfigRsp	CID, OutMTU, InFlow	Result

Description:

This primitive represents the L2CAP_ConfigRsp.

The use of this primitive issues a response to a configuration request event indication. Input parameters include the local CID of the endpoint being configured, outgoing transmit MTU (which may be equal or less to the OutMTU parameter in the L2CA_ConfigInd event) and the accepted flowspec for incoming traffic. The output parameter is the Result value.

Input Parameters:

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Local channel identifier

OutMTU *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Maximum transmission unit this channel will send

InFlow *Type: Flow* *Size: x octets*

Value	Description
FlowSpec	Quality of service parameters dealing with the traffic characteristics of th incoming data flow

Output Parameters:

Result *Size 2 octets*

Value	Description
0x0000	Configuration is successful. Parameters contain agreed upon values
0x0001	Configuration failed – unacceptable parameters
0x0002	Configuration failed – rejected
0x0003	Configuration failed – invalid CID
0x0004	Configuration failed – unknown options
0xXXXX	Reserved

7.6 DISCONNECT

Service	Input Parameters	Output Parameters
L2CA_DisconnectReq	CID	Result

Description:

This primitive represents the L2CAP_DisconnectReq and the returned output parameters represent the corresponding L2CAP_DisconnectRsp or the RTX timer expiration.

The use of this primitive requests the disconnection of the channel. Input parameter is the *CID* representing the local channel endpoint. Output parameter is *Result. Result* is zero if a L2CAP_DisconnectRsp is received, otherwise a non-zero value is returned. Once disconnection has been requested, no process will be able to successfully read or write from the CID. Writes in progress should continue to be processed.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Disconnection successful. This is a result of the receipt of a disconnection response message
0xEEEE	Disconnection timeout occurred.

7.7 WRITE

Service	Input Parameters	Output Parameters
L2CA_DataWrite	CID, Length, OutBuffer	Size, Result

Description:

The use of this primitive requests the transfer of data across the channel. If the length of the data exceeds the OutMTU then only the first OutMTU bytes are sent This command may be used for both connection-oriented and connection-less traffic.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Length *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the buffer where data to be transmitted are stored

OutBuffer *Type: pointer* *Size: N/A*

Value	Description
N/A	Address of the input buffer used to store the message

Output Parameters:

Size *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	The number of bytes transferred

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful write
0x0001	Error – Flush timeout expired
0x0002	Error – Link termination (perhaps this should be left to the indication)

7.8 READ

Service	Input Parameters	Output Parameters
L2CA_DataRead	CID, Length, InBuffer	Result, N

Description:

The use of this primitive requests for the reception of data. This request returns when data is available or the link is terminated. The data returned represents a single L2CAP payload. If not enough data is available, the command will block until the data arrives or the link is terminated. If the payload is bigger than the buffer, only the portion of the payload that fits into the buffer will be returned, and the remainder of the payload will be discarded. This command may be used for both connection-oriented and connectionless traffic.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	CID

Length *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the buffer where received data are to be stored

InBuffer *Type: pointer* *Size: N/A*

Value	Description
N/A	Address of the buffer used to store the message

Output parameters:

Result

Value	Description
0x0000	Success

N *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Number of bytes transferred to InBuffer

7.9 GROUP CREATE

Service	Input Parameters	Output Parameters
L2CA_GroupCreate	PSM	CID

Description:

The use of this primitive requests the creation of a CID to represent a logical connection to multiple devices. Input parameter is the *PSM* value that the outgoing connectionless traffic is labelled with, and the filter used for incoming traffic. Output parameter is the *CID* representing the local endpoint. On creation, the group is empty but incoming traffic destined for the PSM value is readable.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Protocol/service multiplexer value

Output Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

7.10 GROUP CLOSE

Service	Input Parameters	Output Parameters
L2CA_GroupClose	CID	Result

Description:

The use of this primitive closes down a Group.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful closure of the channel
0x0001	Invalid CID

7.11 GROUP ADD MEMBER

Service	Input Parameters	Output Parameters
L2CA_GroupAddMember	CID, BD_ADDR	Result

Description:

The use of this primitive requests the addition of a member to a group. The input parameter includes the CID representing the group and the BD_ADDR of the group member to be added. The output parameter Result confirms the success or failure of the request.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Remote device address

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure to establish connection to remote device
Other	Reserved

7.12 GROUP REMOVE MEMBER

Service	Input Parameters	Output Parameters
L2CA_GroupRemoveMember	CID, BD_ADDR	Result

Description:

The use of this primitive requests the removal of a member from a group. The input parameters include the CID representing the group and BD_ADDR of the group member to be removed. The output parameter Result confirms the success or failure of the request.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address device to be removed

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure – device not a member of the group
Other	Reserved

7.13 GET GROUP MEMBERSHIP

Service	Input Parameters	Output Parameters
L2CA_GroupMembership	CID	Result, N, BD_ADDR_Lst

Description:

The use of this primitive requests a report of the members of a group. The input parameter CID represents the group being queried. The output parameter Result confirms the success or failure of the operation. If the Result is successful, BD_ADDR_Lst is a list of the Bluetooth addresses of the N members of the group.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure – group does not exist
Other	Reserved

N *Type: uint* *Size: 2 octets*

Value	Description
0x0000-0xFFFF	The number of devices in the group identified by the channel end-point CID. If Result indicates failure, N should be set to 0

BD_ADDR_List *Type: pointer* *Size: N/A*

Value	Description
0xXXXXXXXXXXXX	List of N unique Bluetooth addresses of the devices in the group identified by the channel end-point CID. If Result indicates failure, the all-zero address is the only address that should be returned

7.14 PING

Service	Input Parameters:	Output Parameters:
L2CA_Ping	BD_ADDR, ECHO_DATA, Length	Result, ECHO_DATA, Size

Description:

This primitive represents the initiation of an L2CA_EchoReq command and the reception of the corresponding L2CA_EchoRsp command.

Input Parameters:

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device.

ECHO_DATA *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents to be transmitted in the data payload of the Echo Request command.

Length *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the data in the buffer.

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response received.
0x0001	Timeout occurred.

ECHO_DATA *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents received in the data payload of the Echo Response command.

Size *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the data in the buffer.

7.15 GETINFO

Service	Input Parameters	Output Parameters
L2CA_GetInfo	BD_ADDR, InfoType	Result, InfoData, Size

Description:

This primitive represents the initiation of an L2CA_InfoReq command and the reception of the corresponding L2CA_InfoRsp command.

Input Parameters:

BD_ADDR *Type: unit* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device

InfoType *Type: unit* *Size: 2 octets*

Value	Description
0x0001	Maximum connectionless MTU size

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response received
0x0001	Not supported
0x0002	Informational PDU rejected, not supported by remote device
0x0003	Timeout occurred

InfoData *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents received in the data payload of the Information Response command.

Size *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the data in the InfoData buffer.

7.16 DISABLE CONNECTIONLESS TRAFFIC

Service	Input Parameters	Output Parameters
L2CA_DisableCLT	PSM	Result

Description:

General request to disable the reception of connectionless packets. The input parameter is the *PSM* value indicating service that should be blocked. This command may be used to incrementally disable a set of PSM values. The use of the 'invalid' PSM 0x0000 blocks all connectionless traffic. The output parameter *Result* indicates the success or failure of the command. A limited device might support only general blocking rather than PSM-specific blocks and would fail to block a single non-zero PSM value.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Block all connectionless traffic
0xXXXX	Protocol/Service Multiplexer field to be blocked

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful
0x0001	Failure – not supported

7.17 ENABLE CONNECTIONLESS TRAFFIC

Service	Input Parameters	Output Parameters
L2CA_EnableCLT	PSM	Result

Description:

General request to enable the reception of connectionless packets. The input parameter is the *PSM* value indicating the service that should be unblocked. This command may be used to incrementally enable a set of PSM values. The use of the 'invalid' PSM 0x0000 enables all connectionless traffic. The output parameter *Result* indicates the success or failure of the command. A limited device might support only general enabling rather than PSM-specific filters, and would fail to enable a single non-zero PSM value.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Enable all connectionless traffic
0xXXXX	Protocol/Service Multiplexer field to enable

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful
0x0001	Failure – not supported

8 SUMMARY

The Logical Link Control and Adaptation Protocol (L2CAP) is one of two link level protocols running over the Baseband. L2CAP is responsible for higher level protocol multiplexing, MTU abstraction, group management, and conveying quality of service information to the link level.

Protocol multiplexing is supported by defining channels. Each channel is bound to a single protocol in a many-to-one fashion. Multiple channels can be bound to the same protocol, but a channel cannot be bound to multiple protocols. Each L2CAP packet received on a channel is directed to the appropriate higher level protocol.

L2CAP abstracts the variable-sized packets used by the Baseband Protocol (page 33). It supports large packet sizes up to 64 kilobytes using a low-overhead segmentation-and-reassembly mechanism.

Group management provides the abstraction of a group of units allowing more efficient mapping between groups and members of the Bluetooth piconet. Group communication is connectionless and unreliable. When composed of only a pair of units, groups provide connectionless channel alternative to L2CAP's connection-oriented channel.

L2CAP conveys QoS information across channels and provides some admission control to prevent additional channels from violating existing QoS contracts.

9 REFERENCES

[1] Internet Engineering Task Force, "A Proposed Flow Specification", RFC 1363, September 1992.

10 LIST OF FIGURES

11 LIST OF TABLES

TERMS AND ABBREVIATIONS

Baseband	Baseband Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IrDA	Infra-red Data Association
L_CH	Logical Channel
LC	Link Controller
LM	Link Manager
LMP	Link Manager Protocol
MTU	Maximum Transmission Unit
PPP	Point-to-Point Protocol
Reliable	Characteristic of an L2CAP channel that has an infinite flush timeout
RFC	Request For Comments
SAR	Segmentation and Reassembly

APPENDIX A: CONFIGURATION MSCs

The examples in this appendix describe a sample of the multiple possible configuration scenarios that might occur. Currently, these are provided as suggestions and may change in the next update of the Specification.

Figure I illustrates the basic configuration process. In this example, the devices exchange MTU information. All other values are assumed to be default.

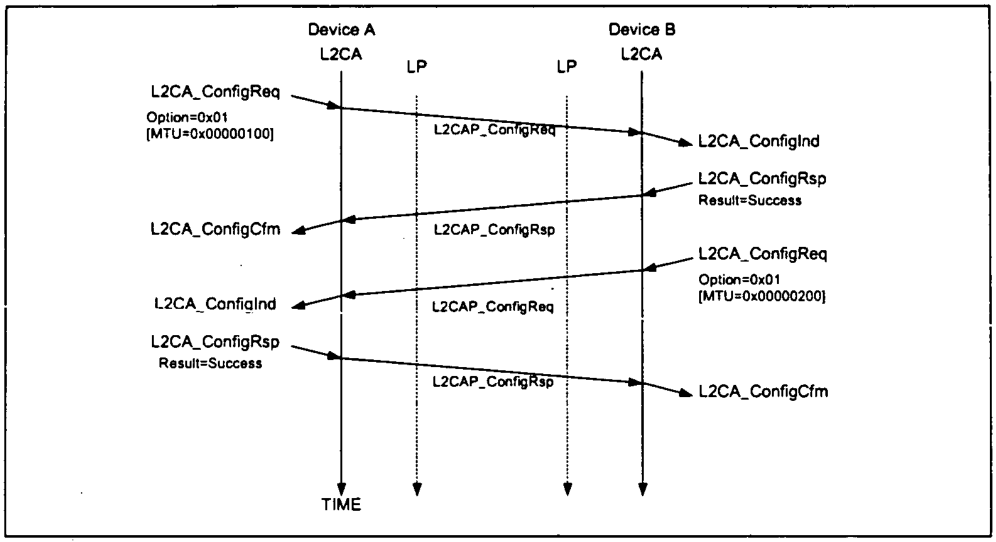


Figure I: Basic MTU exchange

Figure II on page 319 illustrates how two devices interoperate even though one device supports more options than the other does. Device A is an upgraded version. It uses a hypothetically defined option type 0x20 for link-level security. Device B rejects the command using the Configuration Response packet with result 'unknown parameter' informing Device A that option 0x20 is not understood. Device A then resends the request omitting option 0x20. Device B notices that it does not need to such a large MTU and accepts the request but includes in the response the MTU option informing Device A that Device B will not send an L2CAP packet with a payload larger than 0x80 octets over this channel. On receipt of the response, Device A could reduce the buffer allocated to hold incoming traffic.

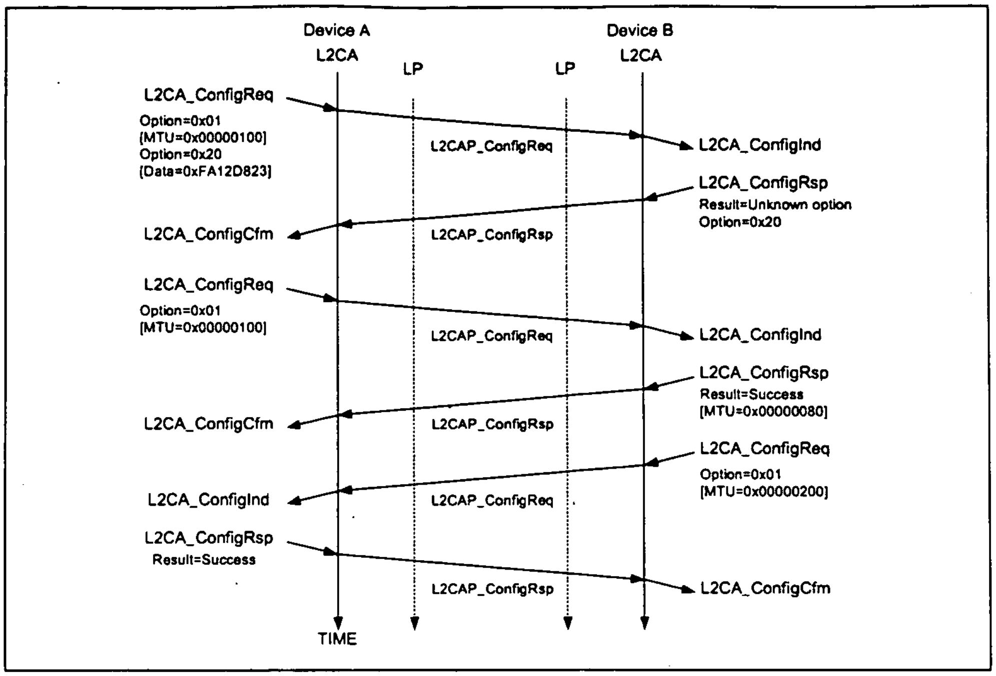


Figure II: Dealing with Unknown Options

Figure III on page 320 illustrates an unsuccessful configuration request. There are two problems described by this example. The first problem is that the configuration request is placed in an L2CAP packet that cannot be accepted by the remote device, due to its size. The remote device informs the sender of this problem using the Command Reject message. Device A then resends the configuration options using two smaller L2CAP_ConfigReq messages.

The second problem is an attempt to configure a channel with an invalid CID. For example device B may not have an open connection on that CID (0x01234567 in this example case).

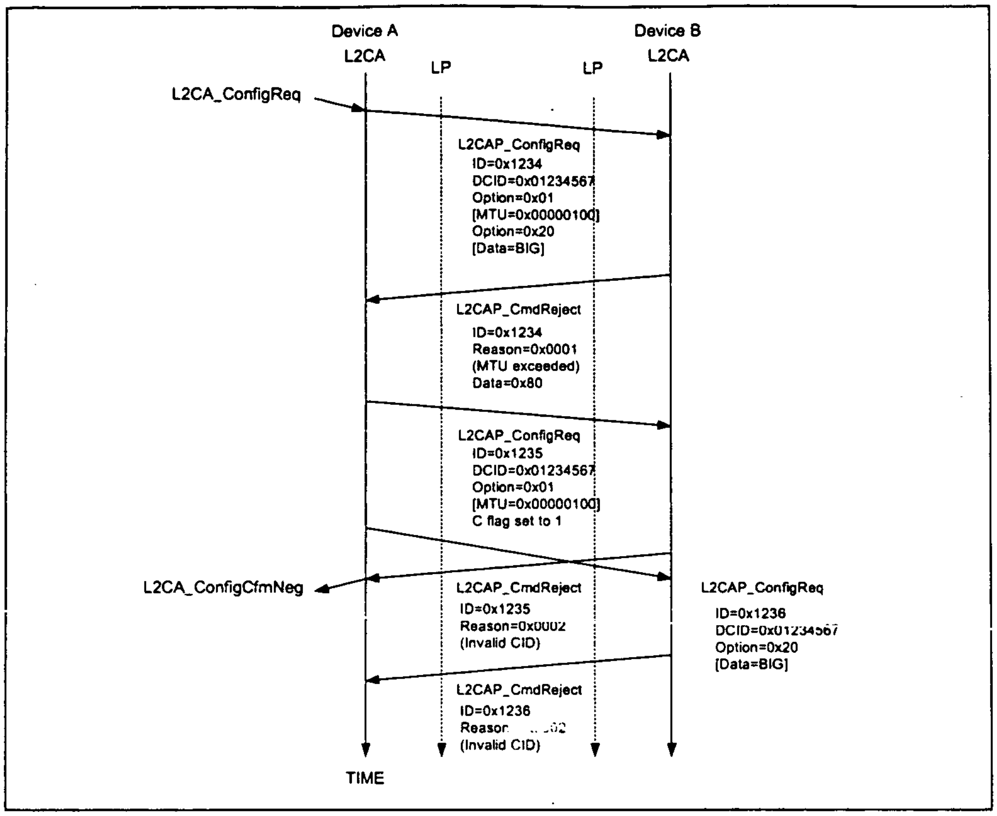


Figure III: Unsuccessful Configuration Request

APPENDIX B: IMPLEMENTATION GUIDELINES

This section contains some guidelines for implementations. These guidelines are not part of the compliance tests. At the moment they are simply suggestions on how to solve some difficult problems.

RTX TIMER

Implementations should not start this timer on an L2CAP Connection Request packet unless the physical link has been established. Otherwise the Baseband paging mechanism might increase the cost of the request beyond that of the minimal timeout value. If an implementation performs some form of security check it is recommended that the connection pending response be sent back prior to any consultation with a security manager that might perform Baseband authentication commands. If any security check requires user interaction, the link might timeout waiting for the user to enter a PIN.

QOS MAPPING TO LM AND L2CAP IMPLEMENTATIONS

Token Rate

The Link Manager (LM) should ensure data is removed from the transmission buffer at this rate. The LM should ensure the polling interval is fast enough to support this data rate. The polling interval should be adjusted if the packet type changes. If the buffer overflows, and the service type is Guaranteed, a QoS violation should be reported. If the service type is Best Effort, and a Token Rate was non-zero, a QoS violation should also be reported.

Given a Token Rate of 0xFFFFFFFF, and Service Type of Guaranteed, the LM should refuse any additional connections from remote devices and disable all periodic scans.

Token Bucket Size

L2CAP implementations should ensure that a buffer meeting the size request is allocated for the channel. If no buffer is available, and the service type is Guaranteed, the request should be rejected. If no appropriately sized buffer is available, and the service type is Best Effort, the largest available buffer should be allocated.

Peak Bandwidth

If the token bucket buffer overflows, a QoS violation should be raised.

Latency

The LM should ensure the polling interval is at least this value. If the polling interval necessary to support the token rate is less than this value, the smaller interval should be used. If this interval cannot be supported, a QoS violation should be raised.

Delay Variation

The LM may ignore this value because there is no clear mapping between L2CAP packet delays and the necessary polling interval without requiring the LM to comprehend the length field in L2CAP packets.

COLLISION TABLES

Current Value	Requested Value	Result
X	X	X
X	Y	If (X < Y) then X, else Y

Table I: Result of Second Link Timeout Request

Current Value	Requested Value	Result
N	0	N
N	N	N
N	M != N	Reject

Table II: Result of Second Flush Timeout Request

Part E

SERVICE DISCOVERY PROTOCOL (SDP)

This specification defines a protocol for locating services provided by or available through a Bluetooth device.

CONTENTS

1 INTRODUCTION

1.1 GENERAL DESCRIPTION

The service discovery protocol (SDP) provides a means for applications to discover which services are available and to determine the characteristics of those available services.

1.2 MOTIVATION

Service Discovery in the Bluetooth environment, where the set of services that are available changes dynamically based on the RF proximity of devices in motion, is qualitatively different from service discovery in traditional network-based environments. The service discovery protocol defined in this specification is intended to address the unique characteristics of the Bluetooth environment. See "Appendix A – Background Information," on page 370, for further information on this topic.

1.3 REQUIREMENTS

The following capabilities have been identified as requirements for version 1.0 of the Service Discovery Protocol.

1. SDP shall provide the ability for clients to search for needed services based on specific attributes of those services.
2. SDP shall permit services to be discovered based on the class of service.
3. SDP shall enable browsing of services without a priori knowledge of the specific characteristics of those services.
4. SDP shall provide the means for the discovery of new services that become available when devices enter RF proximity with a client device as well as when a new service is made available on a device that is in RF proximity with the client device.
5. SDP shall provide a mechanism for determining when a service becomes unavailable when devices leave RF proximity with a client device as well as when a service is made unavailable on a device that is in RF proximity with the client device.
6. SDP shall provide for services, classes of services, and attributes of services to be uniquely identified.
7. SDP shall allow a client on one device to discover a service on another device without consulting a third device.
8. SDP should be suitable for use on devices of limited complexity.
9. SDP shall provide a mechanism to incrementally discover information about the services provided by a device. This is intended to minimize the quantity

of data that must be exchanged in order to determine that a particular service is not needed by a client.

10. SDP should support the caching of service discovery information by intermediary agents to improve the speed or efficiency of the discovery process.
11. SDP should be transport independent.
12. SDP shall function while using L2CAP as its transport protocol.
13. SDP shall permit the discovery and use of services that provide access to other service discovery protocols.
14. SDP shall support the creation and definition of new services without requiring registration with a central authority.

1.4 NON-REQUIREMENTS AND DEFERRED REQUIREMENTS

The Bluetooth SIG recognizes that the following capabilities are related to service discovery. These items are not addressed in SDP version 1.0. However, some may be addressed in future revisions of the specification.

1. SDP 1.0 does not provide access to services. It only provides access to information about services.
2. SDP 1.0 does not provide brokering of services.
3. SDP 1.0 does not provide for negotiation of service parameters.
4. SDP 1.0 does not provide for billing of service use.
5. SDP 1.0 does not provide the means for a client to control or change the operation of a service.
6. SDP 1.0 does not provide an event notification when services, or information about services, become unavailable.
7. SDP 1.0 does not provide an event notification when attributes of services are modified.
8. This specification does not define an application programming interface for SDP.
9. SDP 1.0 does not provide support for service agent functions such as service aggregation or service registration.

1.5 CONVENTIONS

1.5.1 Bit And Byte Ordering Conventions

When multiple bit fields are contained in a single byte and represented in a drawing in this specification, the more significant (high-order) bits are shown toward the left and less significant (low-order) bits toward the right.

Multiple-byte fields are drawn with the more significant bytes toward the left and the less significant bytes toward the right. Multiple-byte fields are transferred in network byte order. See Section 4.1 Transfer Byte Order on page 344.

2 OVERVIEW

2.1 SDP CLIENT-SERVER INTERACTION

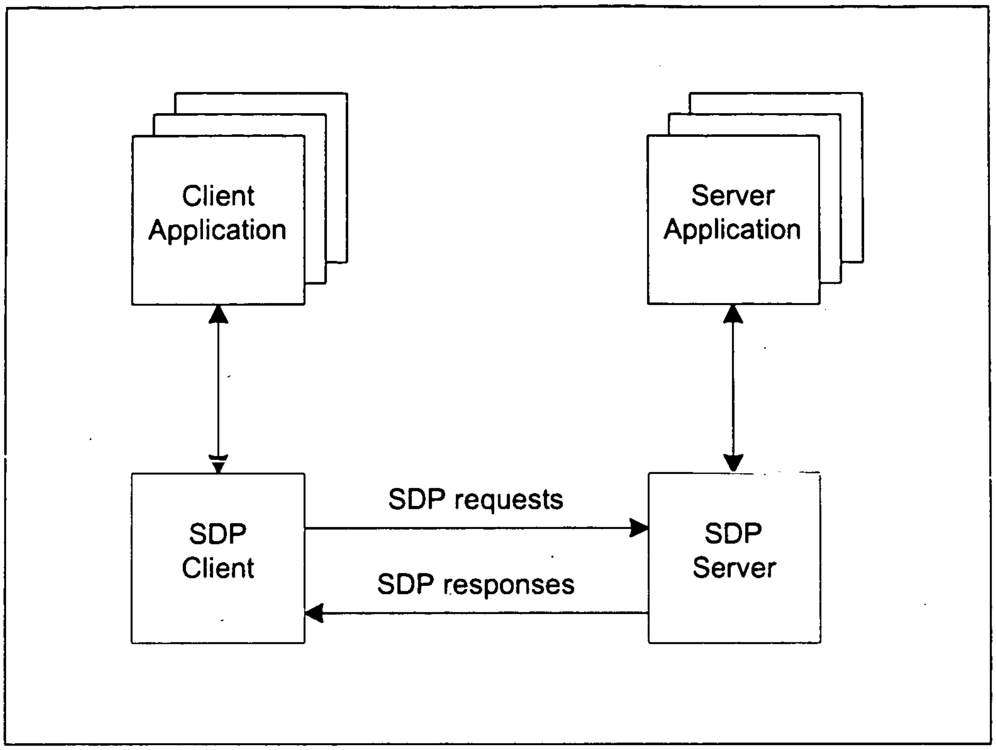


Figure 2.1:

The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

As far as the Service Discovery Protocol (SDP) is concerned, the configuration shown in Figure 1 may be simplified to that shown in Figure 2.

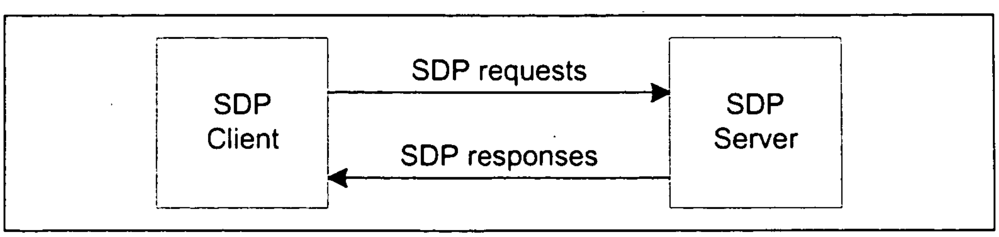


Figure 2.2:

SDP involves communication between an SDP server and an SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client may retrieve information from a service record maintained by the SDP server by issuing an SDP request.

If the client, or an application associated with the client, decides to use a service, it must open a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes (including associated service access protocols), but it does not provide a mechanism for utilizing those services (such as delivering the service access protocols).

There is a maximum of one SDP server per Bluetooth device. (If a Bluetooth device acts only as a client, it needs no SDP server.) A single Bluetooth device may function both as an SDP server and as an SDP client. If multiple applications on a device provide services, an SDP server may act on behalf of those service providers to handle requests for information about the services that they provide.

Similarly, multiple client applications may utilize an SDP client to query servers on behalf of the client applications.

The set of SDP servers that are available to an SDP client can change dynamically based on the RF proximity of the servers to the client. When a server becomes available, a potential client must be notified by a means other than SDP so that the client can use SDP to query the server about its services. Similarly, when a server leaves proximity or becomes unavailable for any reason, there is no explicit notification via the service discovery protocol. However, the client may use SDP to poll the server and may infer that the server is not available if it no longer responds to requests.

Additional information regarding application interaction with SDP is contained in the Bluetooth Service Discovery Profile document.

2.2 SERVICE RECORD

A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software.

All of the information about a service that is maintained by an SDP server is contained within a single service record. The service record consists entirely of a list of service attributes.

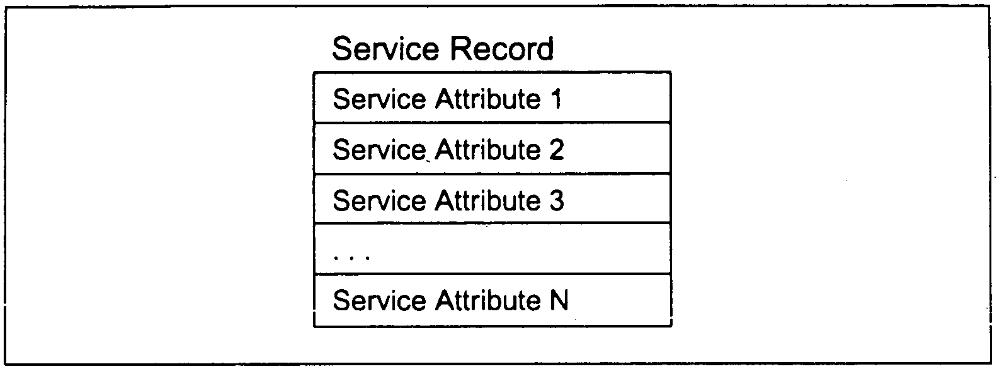


Figure 2.3: Service Record

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will be meaningless if presented to S2.

The service discovery protocol does not provide a mechanism for notifying clients when service records are added to or removed from an SDP server. While an L2CAP (Logical Link Control and Adaptation Protocol) connection is established to a server, a service record handle acquired from the server will remain valid unless the service record it represents is removed. If a service is removed from the server, further requests to the server (during the L2CAP connection in which the service record handle was acquired) using the service's (now stale) record handle will result in an error response indicating an invalid service record handle. An SDP server must ensure that no service record handle values are re-used while an L2CAP connection remains established. Note that service record handles are known to remain valid across successive L2CAP connections while the ServiceDatabaseState attribute value remains unchanged. See the ServiceRecordState and ServiceDatabaseState attributes in Section 5 Service Attribute Definitions on page 358.

There is one service record handle whose meaning is consistent across all SDP servers. This service record handle has the value 0x00000000 and is a

handle to the service record that represents the SDP server itself. This service record contains attributes for the SDP server and the protocol it supports. For example, one of its attributes is the list of SDP protocol versions supported by the server. Service record handle values 0x00000001-0x0000FFFF are reserved.

2.3 SERVICE ATTRIBUTE

Each service attribute describes a single characteristic of a service. Some examples of service attributes are:

ServiceClassIDList	Identifies the type of service represented by a service record. In other words, the list of classes of which the service is an instance
ServiceID	Uniquely identifies a specific instance of a service
ProtocolDescriptorList	Specifies the protocol stack(s) that may be used to utilize a service
ProviderName	The textual name of the individual or organization that provides a service
IconURL	Specifies a URL that refers to an icon image that may be used to represent a service
ServiceName	A text string containing a human-readable name for the service
ServiceDescription	A text string describing the service

See Section 5.1 Universal Attribute Definitions on page 358, for attribute definitions that are common to all service records. Service providers can also define their own service attributes.

A service attribute consists of two components: an attribute ID and an attribute value.

Service Attribute
Attribute ID
Attribute Value

Figure 2.4: Service Attribute

2.4 ATTRIBUTE ID

An attribute ID is a 16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record. The attribute ID also identifies the semantics of the associated attribute value.

A service class definition specifies each of the attribute IDs for a service class and assigns a meaning to the attribute value associated with each attribute ID.

For example, assume that service class C specifies that the attribute value associated with attribute ID 12345 is a text string containing the date the service was created. Assume further that service A is an instance of service class C. If service A's service record contains a service attribute with an attribute ID of 12345, the attribute value must be a text string containing the date that service A was created. However, services that are not instances of service class C may assign a different meaning to attribute ID 12345.

All services belonging to a given service class assign the same meaning to each particular attribute ID. See Section 2.6 Service Class on page 336.

In the Service Discovery Protocol, an attribute ID is often represented as a data element. See Section 3 Data Representation on page 341.

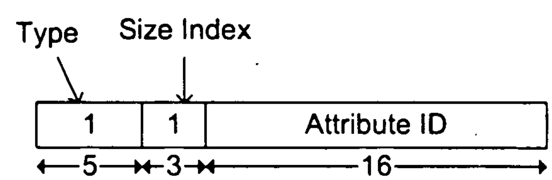


Figure 2.5:

2.5 ATTRIBUTE VALUE

The attribute value is a variable length field whose meaning is determined by the attribute ID associated with it and by the service class of the service record in which the attribute is contained. In the Service Discovery Protocol, an attribute value is represented as a data element. (See Section 3 Data Representation on page 341.) Generally, any type of data element is permitted as an attribute value, subject to the constraints specified in the service class definition that assigns an attribute ID to the attribute and assigns a meaning to the attribute value. See Section 5 Service Attribute Definitions on page 358, for attribute value examples.

2.6 SERVICE CLASS

Each service is an instance of a service class. The service class definition provides the definitions of all attributes contained in service records that represent instances of that class. Each attribute definition specifies the numeric value of the attribute ID, the intended use of the attribute value, and the format of the attribute value. A service record contains attributes that are specific to a service class as well as universal attributes that are common to all services.

Each service class is also assigned a unique identifier. This service class identifier is contained in the attribute value for the ServiceClassIDList attribute, and is represented as a UUID (see Section 2.7.1 UUID on page 337). Since the format and meanings of many attributes in a service record are dependent on the service class of the service record, the ServiceClassIDList attribute is very important. Its value should be examined or verified before any class-specific attributes are used. Since all of the attributes in a service record must conform to all of the service's classes, the service class identifiers contained in the ServiceClassIDList attribute are related. Typically, each service class is a subclass of another class whose identifier is contained in the list. A service subclass definition differs from its superclass in that the subclass contains additional attribute definitions that are specific to the subclass. The service class identifiers in the ServiceClassIDList attribute are listed in order from the most specific class to the most general class.

When a new service class is defined that is a subclass of an existing service class, the new service class retains all of the attributes defined in its superclass. Additional attributes will be defined that are specific to the new service class. In other words, the mechanism for adding new attributes to some of the instances of an existing service class is to create a new service class that is a subclass of the existing service class.

2.6.1 A Printer Service Class Example

A color postscript printer with duplex capability might conform to 4 ServiceClass definitions and have a ServiceClassIDList with UUIDs (See Section 2.7.1 UUID on page 337.) representing the following ServiceClasses:

DuplexColorPostscriptPrinterServiceClassID,
ColorPostscriptPrinterServiceClassID,
PostscriptPrinterServiceClassID,
PrinterServiceClassID

Note that this example is only illustrative. This may not be a practical printer class hierarchy.

2.7 SEARCHING FOR SERVICES

Once an SDP client has a service record handle, it may easily request the values of specific attributes, but how does a client initially acquire a service record handle for the desired service records? The Service Search transaction allows a client to retrieve the service record handles for particular service records based on the values of attributes contained within those service records.

The capability search for service records based on the values of arbitrary attributes is not provided. Rather, the capability is provided to search only for attributes whose values are Universally Unique Identifiers[1] (UUIDs). Important attributes of services that can be used to search for a service are represented as UUIDs.

2.7.1 UUID

A UUID is a universally unique identifier that is guaranteed to be unique across all space and all time. UUIDs can be independently created in a distributed fashion. No central registry of assigned UUIDs is required. A UUID is a 128-bit value.

To reduce the burden of storing and transferring 128-bit UUID values, a range of UUID values has been pre-allocated for assignment to often-used, registered purposes. The first UUID in this pre-allocated range is known as the Bluetooth Base UUID and has the value 00000000-0000-1000-7007-00805F9B34FB, from the Bluetooth Assigned Numbers document. UUID values in the pre-allocated range have aliases that are represented as 16-bit or 32-bit values. These aliases are often called 16-bit and 32-bit UUIDs, but it is important to note that each actually represents a 128-bit UUID value.

The full 128-bit value of a 16-bit or 32-bit UUID may be computed by a simple arithmetic operation.

$$128\_bit\_value = 16\_bit\_value * 2^{96} + Bluetooth\_Base\_UUID$$

$$128\_bit\_value = 32\_bit\_value * 2^{96} + Bluetooth\_Base\_UUID$$

A 16-bit UUID may be converted to 32-bit UUID format by zero-extending the 16-bit value to 32-bits. An equivalent method is to add the 16-bit UUID value to a zero-valued 32-bit UUID.

Note that two 16-bit UUIDs may be compared directly, as may two 32-bit UUIDs or two 128-bit UUIDs. If two UUIDs of differing sizes are to be compared, the shorter UUID must be converted to the longer UUID format before comparison.

1. The format of UUIDs is defined by the International Organization for Standardization in ISO/IEC 11578:1996. "Information technology – Open Systems Interconnection – Remote Procedure Call (RPC)"

2.7.2 Servic Search Patterns

A service search pattern is a list of UUIDs used to locate matching service records. A service search pattern is said to match a service record if each and every UUID in the service search pattern is contained within any of the service record's attribute values. The UUIDs need not be contained within any specific attributes or in any particular order within the service record. The service search pattern matches if the UUIDs it contains constitute a subset of the UUIDs in the service record's attribute values. The only time a service search pattern does not match a service record is if the service search pattern contains at least one UUID that is not contained within the service record's attribute values. Note also that a valid service search pattern must contain at least one UUID.

2.8 BROWSING FOR SERVICES

Normally, a client searches for services based on some desired characteristic(s) (represented by a UUID) of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server's service records without any a priori information about the services. This process of looking for any offered services is termed browsing. In SDP, the mechanism for browsing for services is based on an attribute shared by all service classes. This attribute is called the BrowseGroupList attribute. The value of this attribute contains a list of UUIDs. Each UUID represents a browse group with which a service may be associated for the purpose of browsing.

When a client desires to browse an SDP server's services, it creates a service search pattern containing the UUID that represents the root browse group. All services that may be browsed at the top level are made members of the root browse group by having the root browse group's UUID as a value within the BrowseGroupList attribute.

Normally, if an SDP server has relatively few services, all of its services will be placed in the root browse group. However, the services offered by an SDP server may be organized in a browse group hierarchy, by defining additional browse groups below the root browse group. Each of these additional browse groups is described by a service record with a service class of BrowseGroupDescriptor.

A browse group descriptor service record defines a new browse group by means of its Group ID attribute. In order for a service contained in one of these newly defined browse groups to be browseable, the browse group descriptor service record that defines the new browse group must in turn be browseable. The hierarchy of browseable services that is provided by the use of browse group descriptor service records allows the services contained in an SDP server to be incrementally browsed and is particularly useful when the SDP server contains many service records.

2.8.1 Example S rvic Browsing Hierarchy

Here is a fictitious service browsing hierarchy that may illuminate the manner in which browse group descriptors are used. Browse group descriptor service records are identified with (G); other service records with (S).

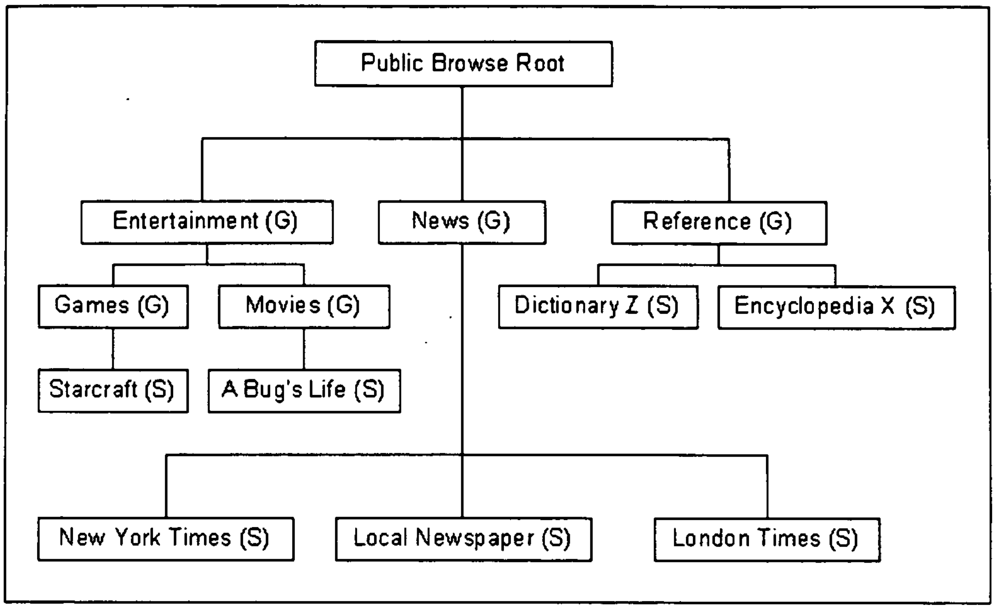


Figure 2.6:

This table shows the services records and service attributes necessary to implement the browse hierarchy.

Service Name	Service Class	Attribute Name	Attribute Value
Entertainment	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	**EntertainmentID**
News	BrowsegroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	**NewsID**
Reference	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	**ReferenceID**
Games	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	**GamesID**
Movies	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	**MoviesID**
Starcraft	Video Game Class ID	BrowseGroupList	GamesID

Table 2.1:

A Bug's Life	Movie Class ID	BrowseGroupList	MovieID
Dictionary Z	Dictionary Class ID	BrowseGroupList	Ref renceID
Encyclopedia X	Encyclopedia Class ID	BrowseGroupList	ReferenceID
New York Times	Newspaper ID	BrowseGroupList	NewspaperID
London Times	Newspaper ID	BrowseGroupList	NewspaperID
Local Newspaper	Newspaper ID	BrowseGroupList	NewspaperID

Table 2.1:

3 DATA REPRESENTATION

Attribute values can contain information of various types with arbitrary complexity; thus enabling an attribute list to be generally useful across a wide variety of service classes and environments.

SDP defines a simple mechanism to describe the data contained within an attribute value. The primitive construct used is the data element.

3.1 DATA ELEMENT

A data element is a typed data representation. It consists of two fields: a header field and a data field. The header field, in turn, is composed of two parts: a type descriptor and a size descriptor. The data is a sequence of bytes whose length is specified in the size descriptor (described in Section 3.3 Data Element Size Descriptor on page 342) and whose meaning is (partially) specified by the type descriptor.

3.2 DATA ELEMENT TYPE DESCRIPTOR

A data element type is represented as a 5-bit type descriptor. The type descriptor is contained in the most significant (high-order) 5 bits of the first byte of the data element header. The following types have been defined.

Type Descriptor Value	Valid Size Descriptor Values	Type Description
0	0	Nil, the null type
1	0, 1, 2, 3, 4	Unsigned Integer
2	0, 1, 2, 3, 4	Signed twos-complement integer
3	1, 2, 4	UUID, a universally unique identifier
4	5, 6, 7	Text string
5	0	Boolean
6	5, 6, 7	Data element sequence, a data element whose data field is a sequence of data elements
7	5, 6, 7	Data element alternative, data element whose data field is a sequence of data elements from which one data element is to be selected.
8	5, 6, 7	URL, a uniform resource locator
9-31		Reserved

Table 3.1:

3.3 DATA ELEMENT SIZE DESCRIPTOR

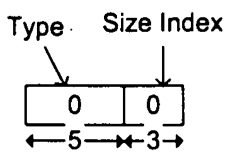
The data element size descriptor is represented as a 3-bit size index followed by 0, 8, 16, or 32 bits. The size index is contained in the least significant (low-order) 3 bits of the first byte of the data element header. The size index is encoded as follows.

Size Index	Additional bits	Data Size
0	0	1 byte. Exception: if the data element type is nil, the data size is 0 bytes.
1	0	2 bytes
2	0	4 bytes
3	0	8 bytes
4	0	16 bytes
5	8	The data size is contained in the additional 8 bits, which are interpreted as an unsigned integer.
6	16	The data size is contained in the additional 16 bits, which are interpreted as an unsigned integer.
7	32	The data size is contained in the additional 32 bits, which are interpreted as an unsigned integer.

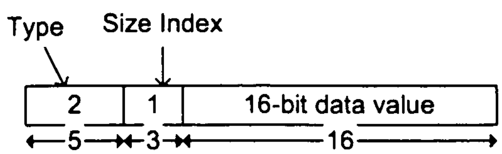
Table 3.2:

3.4 DATA ELEMENT EXAMPLES

Nil is represented as:



A 16-bit signed integer is represented as:



he 3 character ASCII string "Hat" is represented as:

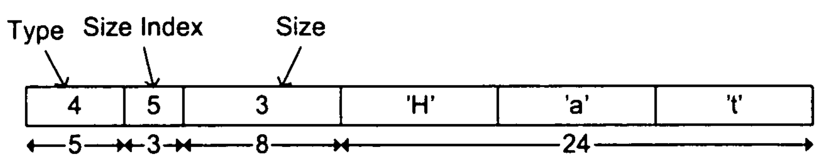


Figure 3.1:

4 PROTOCOL DESCRIPTION

SDP is a simple protocol with minimal requirements on the underlying transport. It can function over a reliable packet transport (or even unreliable, if the client implements timeouts and repeats requests as necessary).

SDP uses a request/response model where each transaction consists of one request protocol data unit (PDU) and one response PDU. However, the requests may potentially be pipelined and responses may potentially be returned out of order.

In the specific case where SDP utilises the Bluetooth L2CAP transport protocol, multiple SDP PDUs may be sent in a single L2CAP packet, but only one L2CAP packet per connection to a given SDP server may be outstanding at a given instant. Limiting SDP to sending one unacknowledged packet provides a simple form of flow control.

The protocol examples found in Appendix B – Example SDP Transactions, may be helpful in understanding the protocol transactions.

4.1 TRANSFER BYTE ORDER

The service discovery protocol transfers multiple-byte fields in standard network byte order (Big Endian), with more significant (high-order) bytes being transferred before less-significant (low-order) bytes.

4.2 PROTOCOL DATA UNIT FORMAT

Every SDP PDU consists of a PDU header followed by PDU-specific parameters. The header contains three fields: a PDU ID, a Transaction ID, and a ParameterLength. Each of these header fields is described here. Parameters may include a continuation state parameter, described below; PDU-specific parameters for each PDU type are described later in separate PDU descriptions.

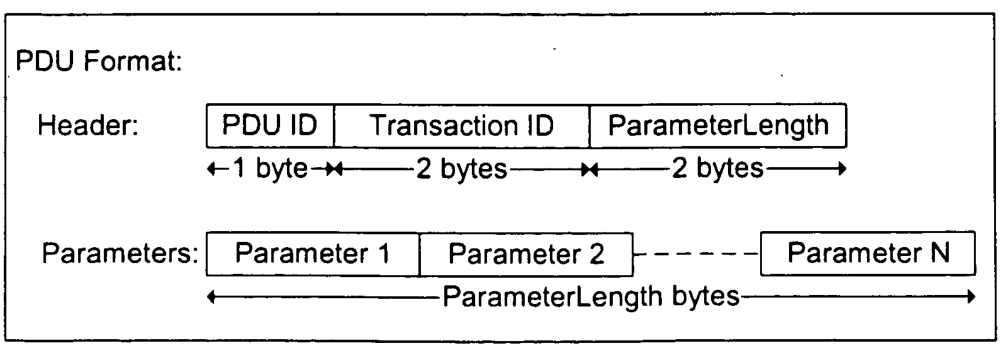


Figure 4.1:

PDU ID: *Size: 1 Byte*

Value	Parameter Description
N	The PDU ID field identifies the type of PDU. I.e. its meaning and the specific parameters.
0x00	Reserved
0x01	SDP_ErrorResponse
0x02	SDP_ServiceSearchRequest
0x03	SDP_ServiceSearchResponse
0x04	SDP_ServiceAttributeRequest
0x05	SDP_ServiceAttributeResponse
0x06	SDP_ServiceSearchAttributeRequest
0x07	SDP_ServiceSearchAttributeResponse
0x07-0xFF	Reserved

TransactionID: *Size: 2 Bytes*

Value	Parameter Description
N	The TransactionID field uniquely identifies request PDUs and is used to match response PDUs to request PDUs. The SDP client can choose any value for a request's TransactionID provided that it is different from all outstanding requests. The TransactionID value in response PDUs is required to be the same as the request that is being responded to. Range: 0x0000 – 0xFFFF

ParameterLength: *Size: 2 Bytes*

Value	Parameter Description
N	The ParameterLength field specifies the length (in bytes) of all parameters contained in the PDU. Range: 0x0000 – 0xFFFF

4.3 PARTIAL RESPONSES AND CONTINUATION STATE

Some SDP requests may require responses that are larger than can fit in a single response PDU. In this case, the SDP server will generate a partial response along with a continuation state parameter. The continuation state parameter can be supplied by the client in a subsequent request to retrieve the next portion of the complete response. The continuation state parameter is a variable length field whose first byte contains the number of additional bytes of continuation information in the field. The format of the continuation information is not standardized among SDP servers. Each continuation state parameter is meaningful only to the SDP server that generated it.

InfoLength	Continuation Information
←1 byte→	←InfoLength bytes→

Figure 4.2: Continuation State Format

After a client receives a partial response and the accompanying continuation state parameter, it can re-issue the original request (with a new transaction ID) and include the continuation state in the new request indicating to the server that the remainder of the original response is desired. The maximum allowable value of the InfoLength field is 16 (0x10).

Note that an SDP server can split a response at any arbitrary boundary when it generates a partial response. The SDP server may select the boundary based on the contents of the reply, but is not required to do so.

4.4 ERROR HANDLING

Each transaction consists of a request and a response PDU. Generally, each type of request PDU has a corresponding type of response PDU. However, if the server determines that a request is improperly formatted or for any reason the server cannot respond with the appropriate PDU type, it will respond with an SDP_ErrorResponse PDU.

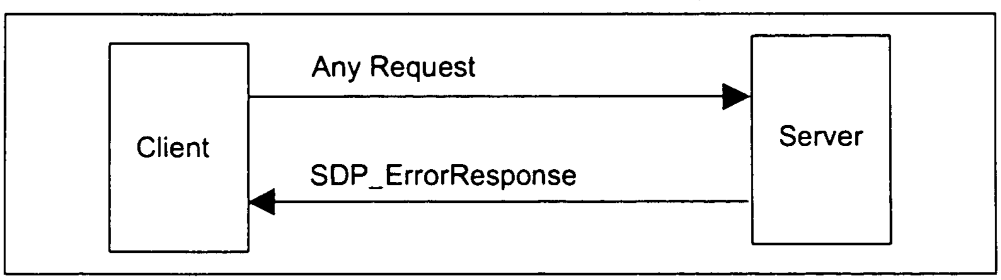


Figure 4.3:

4.4.1 SDP_ErrorRespons PDU

PDU Type	PDU ID	Parameters
SDP_ErrorResponse	0x01	ErrorCode, ErrorInfo

Description:

The SDP server generates this PDU type in response to an improperly formatted request PDU or when the SDP server, for whatever reason, cannot generate an appropriate response PDU.

PDU Parameters:

ErrorCode: *Size: 2 Bytes*

Value	Parameter Description
N	The ErrorCode identifies the reason that an SDP_ErrorResponse PDU was generated.
0x0000	Reserved
0x0001	Invalid/unsupported SDP version
0x0002	Invalid Service Record Handle
0x0003	Invalid request syntax
0x0004	Invalid PDU Size
0x0005	Invalid Continuation State
0x0006	Insufficient Resources to satisfy Request
0x0007-0xFFFF	Reserved

ErrorInfo: *Size: N Bytes*

Value	Parameter Description
Error-specific	ErrorInfo is an ErrorCode-specific parameter. Its interpretation depends on the ErrorCode parameter. The currently defined ErrorCode values do not specify the format of an ErrorInfo field.

4.5 SERVICESEARCH TRANSACTION

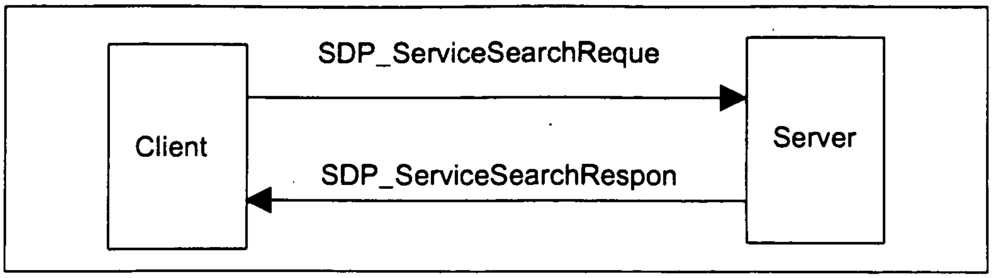


Figure 4.4:

4.5.1 SDP_ServiceSearchRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchRequest	0x02	ServiceSearchPattern, MaximumServiceRecordCount, ContinuationState

Description:

The SDP client generates an SDP_ServiceSearchRequest to locate service records that match the service search pattern given as the first parameter of the PDU. Upon receipt of this request, the SDP server will examine its service record data base and return an SDP_ServiceSearchResponse containing the service record handles of service records that match the given service search pattern.

Note that no mechanism is provided to request information for all service records. However, see Section 2.8 Browsing for Services on page 338 for a description of a mechanism that permits browsing for non-specific services without a priori knowledge of the services.

PDU Parameters:

ServiceSearchPattern: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12[*]. The list of UUIDs constitutes a service search pattern.

*. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

MaximumServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumServiceRecordCount is a 16-bit count specifying the maximum number of service record handles to be returned in the response(s) to this request. The SDP server should not return more handles than this value specifies. If more than N service records match the request, the SDP server determines which matching service record handles to return in the response(s). Range: 0x0001-0xFFFF

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

4.5.2 SDP_ServiceSearchResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchResponse	0x03	TotalServiceRecordCount, CurrentServiceRecordCount, ServiceRecordHandleList, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchResponse upon receipt of a valid SDP_ServiceSearchRequest. The response contains a list of service record handles for service records that match the service search pattern given in the request. Note that if a partial response is generated, it must contain an integral number of complete service record handles; a service record handle value may not be split across multiple PDUs.

PDU Param ters:

TotalServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	The TotalServiceRecordCount is an integer containing the number of service records that match the requested service search pattern. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the MaximumServiceRecordCount value specified in the SDP_ServiceSearchRequest. When multiple partial responses are used, each partial response contains the same value for TotalServiceRecordCount. Range: 0x0000-0xFFFF

CurrentServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	The CurrentServiceRecordCount is an integer indicating the number of service record handles that are contained in the next parameter. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the TotalServiceRecordCount value specified in the current response. Range: 0x0000-0xFFFF

ServiceRecordHandleList: *Size: (CurrentServiceRecordCount*4) Bytes*

Value	Parameter Description
List of 32-bit handles	The ServiceRecordHandleList contains a list of service record handles. The number of handles in the list is given in the CurrentServiceRecordCount parameter. Each of the handles in the list refers to a service record that matches the requested service search pattern. Note that this list of service record handles does not have the format of a data element. It contains no header fields, only the 32-bit service record handles.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is contained in the PDU, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

4.6 SERVICEATTRIBUTE TRANSACTION

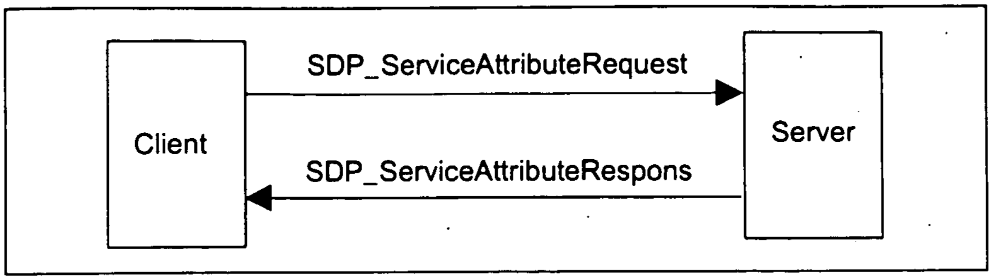


Figure 4.5:

4.6.1 SDP_ServiceAttributeRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeRequest	0x04	ServiceRecordHandle, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP client generates an SDP_ServiceAttributeRequest to retrieve specified attribute values from a specific service record. The service record handle of the desired service record and a list of desired attribute IDs to be retrieved from that service record are supplied as parameters.

Command Parameters:

ServiceRecordHandle: *Size: 4 Bytes*

Value	Parameter Description
32-bit handle	The ServiceRecordHandle parameter specifies the service record from which attribute values are to be retrieved. The handle is obtained via a previous SDP_ServiceSearch transaction.

MaximumAttributeByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response(s) to this request. The SDP server should not return more than N bytes of attribute data in the response(s). If the requested attributes require more than N bytes, the SDP server determines how to truncate th list. Range: 0x0007-0xFFFF